Étude de la dynamique des réseaux biologiques : apprentissage des modèles, intégration des données temporelles et analyse formelle des propriétés dynamiques

Emna Ben Abdallah

UNIVERSITE BRETAGNE LOIRE

CENTRALE NANTES

Thèse de Doctorat

Emna BEN ABDALLAH

Mémoire présenté en vue de l'obtention du
grade de Docteur de l'École centrale de Nantes
sous le sceau de l'Université Bretagne Loire

École doctorale : Mathématiques et STIC

Discipline : Informatique
Unité de recherche : Laboratoire des sciences du Numérique de Nantes (LS2N)

Soutenue le 7 décembre 2017

Étude de la dynamique des réseaux biologiques : apprentissage des modèles, intégration des données temporelles et analyse formelle des propriétés dynamiques

JURY

Président :	**M. Laurent TRILLING**, Professeur des universités, Université Joseph Fourier (TIMC-IMAG)
Rapporteurs :	**M^me^ Hanna KLAUDEL**, Professeur des universités, Université d'Evry-Val d'Essonne (IBISC) **M. Sylvain SENÉ**, Professeur des universités, Université d'Aix-Marseille (LIF)
Examinateur :	**M. Franck DELAUNAY**, Professeur des universités, Université de Nice – Sophia Antipolis (iBV)
Directeur de thèse :	**M. Olivier ROUX**, Professeur des universités, École centrale de Nantes (LS2N)
Co-directeur de thèse :	**M. Morgan MAGNIN**, Professeur des universités, École centrale de Nantes (LS2N)

Remerciements

"Le succès est un chemin que la patience et la persévérance rendent accessible."
Pierre-Simon Ballanche

Je dois continuer, je vais y arriver...
Des problèmes et des échéances stressantes ? Ce n'est pas grave, en tout cas, tout finit par s'arranger...
C'est bien de faire une thèse, non seulement je construis ma carrière professionnelle mais aussi je visite plusieurs nouveaux pays...
J'aurai des beaux résultats publiables...
Fatiguée... mais non rien de grave, repose-toi un peu et tout ira bien...
Il faut juste arriver jusqu'au bout du tunnel...
Ensuite, je vais pouvoir profiter d'une vie normale...
Oui, mes parents seront fières de voir leur fille docteur...
Je dois y arriver...

Des paroles que je n'arrêtais pas de répéter durant la thèse pour garder la motivation. En effet, faire une thèse en étant dépaysée et en vivant toute seule loin de la famille ce n'est pas aussi facile. Une thèse est sans aucun doute un travail de longue haleine, un défi que l'on se donne à soi-même. Mais c'est surtout une formidable histoire de relations, de rencontres et d'amitié. L'achèvement de ce travail mené sur plusieurs années procure une grande satisfaction. Il est l'occasion de se remémorer les différentes embûches qu'il a fallu surmonter mais surtout les personnes qui m'ont permis d'en arriver là.

A l'issue de la rédaction de ma thèse, je suis convaincue que cette dernière est loin d'être un travail solitaire. En effet, je n'aurais jamais pu réaliser ce travail doctoral sans le soutien d'un grand nombre de personnes dont la générosité, la bonne humeur et l'intérêt manifestés à l'égard de ma recherche m'ont permis de progresser dans cette phase délicate. Mais cette période de doctorat aura été probablement l'un des plus beaux chapitres de ma vie. J'aimerais dédier alors ces quelques lignes à remercier ceux et celles qui d'une manière ou d'une autre ont participé à son écriture.

Je tiens tout d'abord à remercier mon directeur de thèse, Olivier Roux, de m'avoir accueillie au sein de son équipe MeForBio. Je lui suis également reconnaissante pour m'avoir fait confiance – malgré notre connaissance plutôt légère que nous avions en décembre 2013– et d'avoir accepté qu'on soumette ensemble une candidature de demande d'une bourse pour le financement de ma thèse au projet Erasmus Mundus.

Je lui suis également reconnaissante pour le temps conséquent qu'il m'a accordé, ses qualités pédagogiques et scientifiques, sa gentillesse et sa sympathie. Je le remercie aussi pour m'avoir guidée, encouragée, conseillée et aussi faite beaucoup voyager en me laissant une grande liberté. J'ai été extrêmement sensible à ses qualités humaines d'écoute et de

compréhension tout au long de ce travail doctoral. Enfin, je le remercie pour toutes ses relectures scrupuleuses du manuscrit et ses suggestions toujours avisées. Je lui adresse mon énorme gratitude pour tout cela.

J'adresse de chaleureux remerciements à mon co-directeur de thèse, Morgan Magnin, pour la confiance qu'il m'a accordée en acceptant d'encadrer ce travail doctoral. J'aimerais également lui dire à quel point j'ai apprécié son attention de tout instant sur mes travaux, ses conseils avisés à la hauteur de ses compétences et de ses réelles qualités humaines et son écoute qui ont été prépondérants pour la bonne réussite de cette thèse. Son énergie et sa confiance ont été des éléments moteurs pour moi. J'ai pris un grand plaisir à travailler avec lui. Je tiens à le remercier aussi pour toutes ses relectures et ses remarques constructives sur ces chapitres. Qu'il soit assuré de ma gratitude.

Mes remerciements vont également à Hanna Klaudel et Sylvain Sené qui ont accepté d'être rapporteurs. Je tiens à remercier également Laurent Trilling et Franck Delaunay pour leur intérêt, leurs très intéressantes discussions et d'avoir accepté de faire partie des membres du jury.

J'adresse de sincères remerciements à Katsumi Inoue, pour m'avoir accueillie chaleureusement au sein de son laboratoire de recherche : *Inoue lab.* du NII de Tokyo dans le cadre d'un stage doctoral. En effet, pendant ce stage, j'ai pu côtoyer un bon nombre de chercheurs de divers pays et avoir pu établir des discussions scientifiques très enrichissantes avec eux. De plus, j'ai eu la chance de visiter plusieurs villes au Japon, de vivre quelques traditions japonaises et surtout de découvrir la cuisine japonaise (à vrai dire j'ai mangé des choses que je n'avais jamais vues auparavant).

En outre, j'adresse mes vifs remerciements à tous ceux sans qui cette thèse ne serait pas ce qu'elle est : aussi bien par les discussions que j'ai eu la chance d'avoir avec eux, leurs suggestions et contributions aux travaux que nous avons pu réaliser ensemble. Je pense ici en particulier au nouveau papa, Maxime Folschette, qui m'a encadrée pendant mon stage de master et avec qui j'ai continué à collaborer pendant ma thèse. Je tiens également à remercier Tony Ribeiro pour sa collaboration, les discussions constructives qu'on a eues et ses conseils même s'ils ne sont pas toujours très pertinents.

Ce travail n'aurait pu être mené à bien sans l'aide du financeur principal de cette thèse qui est le projet Erasmus Mundus EU-METALIC II. J'adresse alors aux responsables de ce projet mes sincères remerciements. Et je remercie aussi l'ensemble du bureau des relations internationales à l'École Centrale de Nantes pour leur disponibilité et leur réactivité si rapide. Je m'adresse principalement à Fouad Bennis, Cyrielle Rohart, Adele Pruvost.

Je tiens aussi à mentionner le plaisir que j'ai eu à travailler au sein du laboratoire LS2N, et j'en remercie ici tous les membres. Un merci très chaleureux à Virginie Dupont et à Isabelle Favreau pour leur bonne humeur et de m'avoir facilité un bon nombre de démarches administratives. Merci à Vincent Tourre, Guillaume Moreau, Jean-Yves Martin et Myriam Servières pour leurs agréables échanges et conseils sur l'enseignement et la recherche.

Je tiens également à remercier tous mes amis et collègues de l'équipe MeForBio, Louis Fippo-Fitime, Bertrand Miannay et Xinwei Chai pour leur aide, leur soutien et leur gentillesse. Les rapports humains dont j'ai profité à leur côté ont fait naître de réels liens d'amitiés qui à mes yeux n'ont pas de prix. Qu'ils soient tous assurés de ma plus profonde gratitude et estime.

Je passe ensuite une dédicace spéciale à tous les jeunes gens que j'ai eu le plaisir de côtoyer durant ces quelques années à Nantes, à savoir mes collègues thésards, mes voisins, et mes amis fréquentés aux événements associatifs.... J'ai eu du plaisir à partager des bons moment avec eux. Qu'ils soient tous assurés de ma sincère et profonde amitié.

Enfin, les mots les plus simples étant les plus forts, je les adresse alors à ma famille. Je remercie mon frère, ma sœur ainsi que leurs compagnons pour leurs encouragements. Je tiens surtout à adresser toute mon affection à mes chers parents. Malgré mon éloignement depuis quelques années, leur confiance, leur tendresse et leur amour me portent et me guident tous les jours. Merci pour avoir fait de moi ce que je suis aujourd'hui. Un grand merci également à mon chéri qui a toujours été là pour moi. Son soutien inconditionnel, son support et son amour ont été d'une grande aide. Est-ce un bon endroit pour dire ce genre de choses ? Je n'en connais en tous cas pas de mauvais. Je vous aime.

Table des matières

Chapitre 1

Introduction

1.1 Contexte et motivations

Un système dynamique est composé d'un ensemble d'entités en interaction. Du fait même de ces interactions, la valeur des grandeurs attachées à ces entités évoluent dans le temps. C'est en étudiant l'évolution de ces valeurs que nous cherchons à comprendre et à *prédire* le comportement de ces systèmes. Un grand intérêt de ces prédictions est de permettre d'envisager de modifier certaines données du système ou de son environnement pour faire en sorte de vérifier de nouvelles propriétés des comportements. Cela repose sur la représentation mathématique des systèmes dynamiques à l'aide d'un *modèle*. Nous sommes intéressés, en particulier, à caractériser l'évolution, au cours du temps, de l'état de tels modèles dynamiques.

La grandeur des entités dont la valeur évolue est une caractéristique dépendante de la nature du système dynamique à modéliser ; par exemple, en économie, cette grandeur peut concerner des montants de capital ou des flux financiers et en chimie, elle peut porter sur les valeurs des concentrations des molécules en présence. Le champ d'application considéré dans cette thèse est principalement la biologie des systèmes (la biologie cellulaire). Ainsi, la grandeur que nous étudions est essentiellement la concentration des composants cellulaires (gènes, ADN, protéines...).

En effet, cette thèse entre dans la perspective de la compréhension de la nature du fonctionnement cellulaire ou plus largement des systèmes biologiques vivants. Il est donc essentiel d'étudier non seulement les propriétés dynamiques individuelles de ses composants cellulaires, mais aussi leurs interactions car les activités et les expressions des composants cellulaires ne sont pas isolées mais dépendantes les unes des autres. En outre, le comportement et le fonctionnement des cellules émergent de ces interactions entre ses composants cellulaires.

Au cours des dernières décennies, l'émergence d'une large gamme de nouvelles technologies a permis de produire une quantité massive de données biologiques (génomique, protéomique...). De plus, avec le développement des nouvelles approches expérimentales, telles que le "*DNA microarrays*" pour la biologie cellulaire, une grande quantité de "*données de séries temporelles*" est maintenant produite tous les jours. Ces données décrivent les niveaux d'expression (i.e., les concentrations) des composants du système biologique étudié au cours du temps et à des instants précis.

Toutes les données nouvellement produites peuvent donner des nouvelles interprétations sur le comportement du système biologique auquel elles se rapportent. Cela conduit à un impact considérable dans le domaine de la *bioinformatique* qui peut aussi tirer profit de ces quantités de données. Ceci justifie alors notre motivation pour le développement des méthodes efficaces pour l'*inférence* (ou *l'apprentissage*) des Réseaux de Régulation Biologique (RRB) modélisant le système biologique étudié à partir des données de séries temporelles.

Dans ce contexte, la thèse que nous présentons a pour but de contribuer à une meilleure compréhension de la biologie des systèmes. Elle propose de s'atteler à la *modélisation*, à l'*inférence* des modèles à partir des données de séries temporelles et à l'*analyse* de ces modèles. Notre objectif est en effet de développer des méthodes originales pour déchiffrer la complexité des systèmes biologiques (et en particulier des RRB qui tiennent compte des propriétés *qualitatives*).

Il est à savoir que le principal défi posé par l'étude des systèmes dynamiques, qu'ils soient biologiques ou non, repose dans la modélisation qui en est faite. Commençons alors par préciser ce qu'est une modélisation d'un système dynamique.

La modélisation est une tentative de décrire, de manière précise, une compréhension des entités du système étudié. Elle englobe les états des entités, leurs interactions les unes avec les autres et les évolutions des comportements qui découlent de ces interactions.

Lorsqu'on commence une démarche de modélisation, le premier problème à aborder est de déterminer exactement quelles caractéristiques inclure dans le modèle, et en particulier, le niveau de détail que le modèle est destiné à capturer. Le modèle doit être suffisamment détaillé et précis afin qu'il puisse, en principe, être utilisé pour *simuler* sur un ordinateur le comportement du système qu'il modélise.

Ainsi, l'art de construire un bon modèle est de capturer les caractéristiques essentielles du système à modéliser sans encombrer le modèle avec des détails non essentiels. En effet, chaque modèle est, dans une certaine mesure, une simplification de la réalité. Mais le modèle est précieux car il prend des idées qui auraient pu être exprimées verbalement ou schématiquement avec des graphes et les rendre plus explicites par des abstractions qui dépendent du formalisme choisi pour modéliser : *quantitatives* ou *qualitatives* ou encore *hybrides*.

Puisque nous nous intéressons principalement au domaine de la biologie cellulaire moléculaire, pour appliquer notre méthode d'inférence des modèles, il est à savoir que de tels modèles peuvent décrire certains mécanismes impliqués dans la traduction de la transcription, la régulation des gènes, les détériorations (mutations) et/ou les processus de réparation d'ADN, le cycle cellulaire ou la signalisation cellulaire. À un niveau supérieur, la modélisation peut être utilisée pour décrire le fonctionnement d'un tissu, d'un organe ou même d'un organisme entier. Ce type de modèles, illustrant une réalité biologique, pourrait avoir des applications pour la recherche des médicaments, par exemple sur le cancer et les thérapies géniques.

Une fois que nous réussissons à obtenir un modèle assez détaillé d'un système, il est possible de "*tester*" si sa compréhension est correcte, en voyant si les implications de leurs modèles sont compatibles avec les données expérimentales observées. En pratique, cette étape de "*validation*" du modèle appris est au cœur de l'approche biologique des systèmes.

En effet, nous adoptons aussi cette démarche pour la validation des modèles appris par notre nouvelle méthode d'inférence introduite dans cette thèse.

Cependant, la modélisation ne représente que la partie initiale pour l'étude des systèmes dynamiques (en l'occurrence les systèmes biologiques). L'objectif principal, dès qu'un modèle satisfaisant est obtenu, est de réussir à *analyser* et à *prédire* la dynamique du système modélisé sur un ordinateur (voir la figure 1.3 en page 20). En effet, la *simulation* et l'*analyse* par ordinateur ont souvent été proposées pour augmenter considérablement l'efficacité de la découverte des médicaments par exemple. À l'heure actuelle, les prédictions faites dans le domaine pharmaceutique ont été utilisées avec un certain succès.

En fait, un modèle rend possible la réalisation d'"*expériences virtuelles*" qui pourraient être difficiles, longues, coûteuses voire impossibles à faire avec le système réel dans le laboratoire. Ainsi, les modèles peuvent également être extrêmement utiles pour induire la *conception* et l'*analyse* d'expériences biologiques complexes. De telles expériences peuvent révéler des relations importantes et qui sont difficiles à prévoir autrement ; comme par exemple, des interactions indirectes entre les composants du modèle ou même des nouvelles propriétés comportementales de la dynamique du système. *C'est dans cette perspective qu'une autre contribution importante est introduite dans cette thèse et qui consiste à analyser formellement certaines propriétés dynamiques des RRB.*

Nous développons dans ce manuscrit une analyse dynamique qui vérifie dans un RRB, la propriété d'*atteignabilité* d'un objectif (un ensemble de composants) à partir d'un état initial du réseau. Notre méthode est une approche logique et exhaustive qui se montre efficace pour faire face à la complexité de cette problématique. De plus, nous proposons une approche optimale qui retourne le plus court chemin en un temps restreint pour des grands modèles (plus de 50 composants).

En outre, une autre propriété dynamique est étudiée dans cette thèse : c'est celle qui identifie dans un RRB ses *attracteurs* (ensemble d'états dont la dynamique ne peut pas s'échapper). Nous introduisons ainsi dans ce manuscrit une méthode innovante qui analyse dynamiquement les RRB afin d'identifier leurs *attracteurs*. Nous montrons que cette méthode est efficace pour prendre en compte l'énorme complexité d'une telle analyse. Elle réussit à identifier en des temps réduits les attracteurs dans des grands, voire très grands modèles (plus de 100 composants).

L'un des intérêts de l'identification des attracteurs d'un RRB est l'étude de l'effet d'une maladie ou d'une mutation sur un organisme. En effet, cette étude nécessite de trouver les attracteurs dans le modèle pour comprendre les comportements à long terme du système qu'il modélise. D'autre part, de telles propriétés dynamiques sont considérées comme une réponse caractéristique du système modélisé.

En résumé, la modélisation et la simulation par ordinateur sont de plus en plus importantes dans la biologie post-génomique pour intégrer les connaissances et les données expérimentales et faire des prédictions vérifiables sur le comportement de systèmes biologiques complexes. C'est ce qui constitue la motivation principale de cette thèse.

Nos résultats reposent sur le formalisme des *réseaux d'automates* (*Automata Networks*, AN) (Folschette, Paulevé, Magnin & Roux, 2015). Ce formalisme se veut simple et efficace par la modélisation *qualitative* de systèmes dynamiques complexes de grande taille. En particulier, les AN peuvent modéliser les RRB avec différents niveaux d'abstraction dans leur

spécification, apportant ainsi une nouvelle souplesse pour leur étude. La structure simple des AN permet alors de dériver efficacement des propriétés sur les dynamiques produites.

Pour avoir une meilleure prédiction et des bons résultats, nous proposons dans cette thèse un raffinement de la modélisation des RRB représentés avec le formalisme des AN par l'intégration d'une *composante temporelle*. Nous appelons ce formalisme le *réseau d'automates avec le temps* (*Timed Automata Network*, T-AN)

En effet, en simulant un AN, nous construisons une séquence d'événements discrets (i.e., des états du réseau) qui illustre la dynamique du système modélisé sans aucune information sur la durée qui sépare ces événements. Elle ne fournit que des informations sur la chronologie entre les états et rien sur la mesure chronométrique du temps passé dans ces états.

Nous intégrons alors une composante temporelle, appelée "*délai*", dans les interactions qui existent entre les composants du modèle. Chaque délai est spécifique pour une interaction donnée. Et il représente le temps nécessaire pour que cette interaction s'exécute.

Nous montrons dans cette thèse comment nous inférons ce délai à partir des données de séries temporelles. D'autre part, nous introduisons une nouvelle sémantique pour la dynamique des T-AN qui garantit le raffinement de la dynamique du modèle par rapport à celle calculée avec les AN. Nous indiquons que les RRB inférés par notre méthode sont représentés avec le formalisme des T-AN.

La section 1.2 suivante présente une brève introduction aux RRB à partir desquels nos travaux ont des développements. La section 1.3 expose les différentes contributions de cette thèse. Ensuite, la section 1.4 présente le plan général de ce manuscrit de thèse. Nous indiquons que les travaux défendus dans cette thèse sont réalisés dans un cadre de collaborations sur lesquelles nous donnons des précisions dans la section 1.5. Enfin, la section 1.6 fournit la liste des principales notations mathématiques utilisées dans ce manuscrit.

1.2 Réseaux de régulation biologique (RRB)

Cette section présente brièvement les aspects clés des phénomènes de la régulation transcriptionnelle et post-transcriptionnelle dans les cellules qui jouent un rôle crucial dans plusieurs systèmes biologiques. Nous examinons quelques aspects de l'expression et de la régulation des gènes.

En biologie cellulaire, le terme *facteur de transcription* est utilisé pour désigner les catalyseurs (i.e., activateurs) et les répresseurs (i.e., inhibiteurs) des transcriptions spécifiques qui activent ou répriment la transcription des gènes cibles par liaison spécifique aux régions des promoteurs. La figure 1.1 ci-dessous illustre le processus de l'expression génique. En effet, le gène qui se trouve sur l'ADN va subir une transcription dans le noyau pour avoir l'*ARNm* et puis la traduction de ce dernier dans le cytoplasme afin de produire la *protéine*. En fait, dans le système de régulation biologique, c'est la protéine qui va agir et avoir un effet d'activation ou d'inhibition sur un autre processus génique (ou même sur sa propre production). Les boucles de rétroaction impliquant une inhibition ou une stimulation de facteurs de transcription peuvent contrôler les réponses complexes d'expression des gènes et des processus de développement.

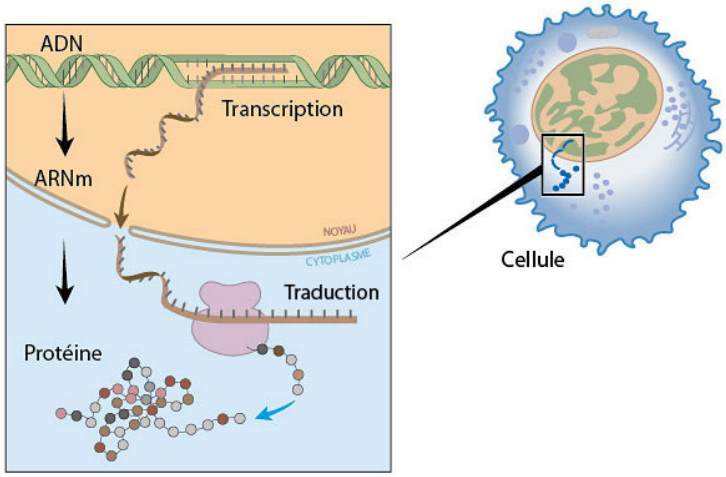


Figure 1.1 : Un schéma qui représente à l'échelle microscopique une cellule. Plus précisément, il illustre le passage de l'information génique du noyau vers la cellule par la transcription de l'ADN en ARN messager (ARNm) et qui sera traduit par la suite en une protéine. Cette figure est prise de (Röhl, 2006).

Ces informations sur l'influence positive ou négative entre les composants d'un système biologique sont typiquement des informations *qualitatives*. Toutes ces informations qualitatives peuvent être englobées dans un *graphe des interactions* : les nœuds représentent une abstraction des entités biologiques (gène, ARNm, etc.) ou une abstraction de plusieurs entités (un complexe biologique), et sont reliés entre eux par des arcs orientés positifs ou négatifs dénotant respectivement une activation ou une inhibition. Nous donnons un exemple de graphe des interactions dans la figure 1.2 ci-dessous (à gauche).

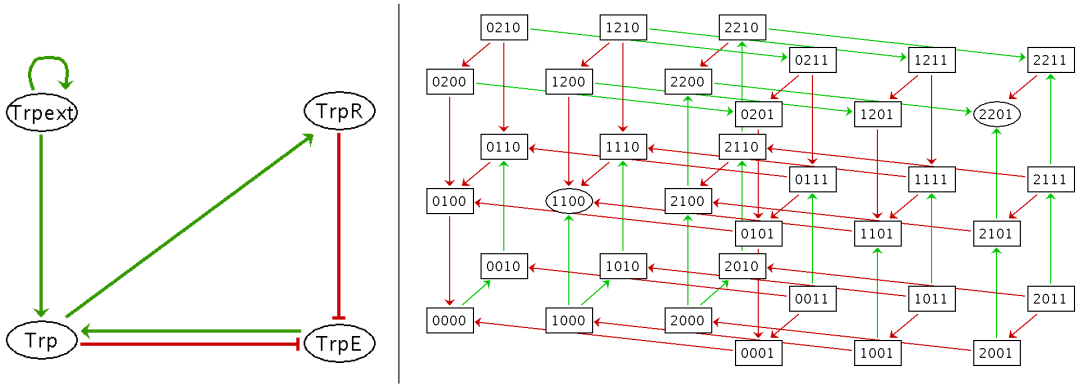


Figure 1.2 : (gauche) Graphe des interactions modélisant les voies métaboliques régulées de la biosynthèse du tryptophane de E. coli (Simao et al., 2005). Chaque nœud représente un composant biologique étiqueté par l'abréviation de son nom. Les arcs en vert représentent une régulation positive (une activation) et ceux en rouge représentent une régulation négative (une inhibition). (droite) Le graphe de transition d'états représentant la dynamique discrète obtenue à partir du graphe des interactions du modèle à gauche généré par l'outil GinSim (Chaouiya et al., 2012). Les arcs en vert (resp. en rouge) représentent une activation (resp. inhibition) d'un composant par le passage d'un niveau vers un niveau supérieur (resp. inférieur). Les niveaux 0, 1, 2 de chacun des composants présentent respectivement qu'il n'est pas exprimé, moyennement exprimé et fortement exprimé. L'ordre du quadruplet est : Trpext, Trp, TrpE et TrpR.

La partie à gauche de la figure 1.2 ci-dessus présente un exemple d'un petit graphe des interactions entre les 4 composants (Trpext, Trp, TrpE et TrpR). En fait, ce graphe n'est qu'une abstraction du RRB réel. Si cette simplification paraît extrême par rapport à la complexité de la réalité biologique qu'elle représente, elle offre néanmoins une certaine compréhension du fonctionnement du système. En particulier, les nœuds du graphe, sous leur forme la plus générale, représentent des gènes, des ARNm et des protéines, et les arcs symbolisent l'interaction ou la régulation. Compte tenu de la multitude de composants disponibles dans une cellule, nous pouvons traiter des graphes d'interactions extrêmement compliqués. Ainsi, déchiffrer la structure et la dynamique de ces réseaux est la première étape vers la compréhension du comportement global des systèmes biologiques.

Les approches de modélisation dynamique sont divisées en deux catégories majeures, à savoir *continues* ou *discrètes*, selon la description des états des nœuds. Les modèles continus sont généralement décrits comme un ensemble d'équations différentielles. Leur avantage est qu'elles permettent de suivre l'évolution de l'expression de chaque composant en continu. Cependant, l'utilisation de ces modèles est entravée par la rareté des détails cinétiques des interactions dans la plupart des cas et par le fait que la résolution de telles équations est très coûteuse et généralement impossible analytiquement.

On trouve les modèles discrets offrant une description qualitative, tels que les modèles booléens e.g., (Thomas & d'Ari, 1990; Kauffman, 1993), les modèles logiques d'états finis e.g., (Sanchez & Thieffry, 2001), et les réseaux de Petri e.g., (Giavitto, Klaudel & Pommereau, 2010; Chaouiya, Klaudel & Pommereau, 2011), etc, qui ne requièrent aucun ou peu de paramètres.

La classe des équations différentielles linéaires par morceau (modèles hybrides) (Kauffman, 1969; Gebert, Radde & Weber, 2007), etc. comble l'écart entre les modèles continus et discrets en caractérisant chaque nœud par deux variables : une concentration continue et une activité discrète. Ces modèles fondent la description logique des RRB avec une désintégration de la concentration linéaire.

Ainsi, le modèle à utiliser parmi ceux introduits ci-dessus, dépend du niveau des détails quantitatifs dans les données expérimentales disponibles. En effet, les modèles continus peuvent être utilisés lorsqu'une information cinétique suffisante est disponible ; en revanche, les modèles discrets sont les mieux adaptés aux systèmes moins riches en détails cinétiques. Les modèles hybrides peuvent être utilisés lorsque des informations partielles sur les paramètres cinétiques sont disponibles.

L'accent est mis dans cette thèse sur les modèles discrets, pour la modélisation des RRB qui ont d'abord été introduits par Kauffman (Kauffman, 1969) et Thomas (Thomas, 1973). Dans ces modèles, chaque nœud peut avoir un nombre fini de niveaux qualitatifs discrets ; par exemple dans les modèles booléens, chaque composant a 2 niveaux : 0 (inactif) et 1 (actif). L'état 0 correspond à une concentration ou une activité inférieure à un certain seuil, qui est insuffisant pour initier le processus ou la régulation prévue, et l'état 1 est une concentration ou une activité supérieure au seuil.

Bien que la concentration des composants dans les systèmes biologiques change continuellement au fil du temps, le profil d'expression d'un composant par rapport à celui de son régulateur est de nature sigmoïdal. Ainsi, ses niveaux d'expression peuvent être bien approchés par des fonctions en escalier (Thomas & d'Ari, 1990). Ceci fournit alors une justification suffisante pour l'utilisation de modèles booléens. Dans le cas des modèles multi-valués (3 niveaux d'expression discrets ou plus), cette approche a l'avantage de

permettre d'abstraire les valeurs des seuils de concentration, qui ne sont pas toujours bien connues mais qui permettent de représenter le niveau de concentration à partir duquel commence la régulation effective d'un composant sur un autre. Ainsi, à chaque niveau d'expression discret d'un composant est associé un ensemble de régulations effectuées sur d'autres composants du modèle.

Finalement, afin d'avoir un raffinement de cette modélisation discrète et de produire plus de précisions sur la dynamique des RRB, il y a eu des enrichissements de ces modélisations par l'intégration d'une dimension temporelle (délai, probabilité). Par l'ajout de cette dimension temporelle (Thomas, 1978; Ahmad, Bernot, Comet, Lime & Roux, 2006; Siebert & Bockmayr, 2006), etc., l'aspect quantitatif est apparu dans ces modèles et des nouveaux comportements sont apparus. Ils expriment alors l'importance de certains paramètres temporels ou stochastiques sur l'évolution de la dynamique du système. Ceci est aussi le cas pour les T-AN introduits dans cette thèse (nous en donnons plus de détails dans le chapitre 3).

1.3 Contributions

Les apports de cette thèse se déclinent en trois contributions principales qui sont énumérées brièvement ci-dessous dans **(A), (B)** et **(C)**. Nous les introduisons avec plus de détails afin d'insister sur leurs apports. Nous situons aussi ces contributions dans la figure 1.3 ci-après qui montre une idée globale sur la motivation de ces travaux.

- **(A)** *L'intégration des paramètres temporels dans les RRB modélisés en AN : vers les T-AN.*
- **(B)** *L'inférence des RRB (modélisés en T-AN) à partir des données de séries temporelles.*
- **(C)** *L'analyse formelle et la vérification exhaustive des propriétés dynamiques dans les RRB : la vérification de l'atteignabilité et l'identification des attracteurs.*

(A) Intégration des paramètres temporels dans les RRB modélisés en AN : vers les T-AN

Dans le but d'introduire une dimension temporelle précise dans les AN modélisant les RRB, nous présentons un nouveau formalisme appelé T-AN. La particularité de notre modélisation via les T-AN est le raffinement qu'elle apporte à la dynamique des AN qui modélisent les RRB. En effet, par rapport à un AN, dans un T-AN, les interactions entre les composants sont enrichies par des paramètres temporels appelés *les délais* ; chaque délai représente la période temporelle nécessaire pour que la régulation d'un composant par un autre s'effectue. Autrement dit, c'est la durée entre l'instant auquel l'interaction commence son activité et l'instant auquel l'effet est complètement produit.

Pour certains systèmes, il est crucial d'avoir une information sur ce temps mis entre deux événements (e.g., le système de l'horloge circadienne). Ainsi, nous étudions l'impact de cet enrichissement sur la dynamique du modèle. En effet, cet enrichissement peut créer de nouvelles évolutions de la dynamique comme il peut en supprimer d'autres. Au cours de l'activation d'une interaction, il pourrait se créer des conflits entre d'autres interactions du modèle sur des ressources partagées (éléments nécessaires pour qu'une interaction se produise). Ainsi, nous présentons une nouvelle sémantique de la dynamique des T-AN que nous adoptons pour affiner la dynamique générale des RRB. Ceci permet alors de capturer

un comportement plus réaliste et donc plus proche du comportement du système biologique modélisé.

(B) Inférence des T-AN à partir des données de séries temporelles

Dans le but d'automatiser le processus de modélisation des RRB, nous proposons une nouvelle approche qui apprend un RRB (modélisé en T-AN) qui est aussi cohérent que possible par rapport à la dynamique observée du système modélisé. Ces observations de la dynamique enregistrent l'évolution des niveaux d'expression des composants du système fournis au cours du temps et elles sont appelées par des *données de séries temporelles*.

La particularité de notre méthode repose principalement sur : (i) l'intégration directe des délais dans le processus d'inférence ; (ii) l'identification des niveaux d'expression discrets des composants qui participent à chaque interaction du réseau ; (iii) et aussi l'identification du signe de cette interaction (activation ou inhibition).

De plus, nous proposons une extension de cette méthode d'inférence qui permet de *réviser* un modèle T-AN existant s'il n'est pas cohérent avec des données de séries temporelles nouvellement fournies (c'est-à-dire différentes de celles à partir desquelles le T-AN à réviser est appris). Ceci entre aussi dans le cadre du processus de la *validation* d'un modèle T-AN après son apprentissage. En effet, si un modèle n'est pas cohérent avec toutes les observations du système, qu'elles englobent ou non des perturbations (e.g., suppression d'un composant), alors il doit être révisé. Cette révision consiste par exemple à ajouter des interactions manquantes.

Nous proposons un raffinement des modèles T-AN appris par l'application d'un ensemble de *filtres* qui permettent, entre autres, de réviser les T-AN appris. En effet, les filtres introduits dans cette thèse assurent un meilleur résultat vis-à-vis des T-AN appris. Parmi ces filtres, il y en a un qui s'occupe d'enlever les incohérences, si elles existent, entre la dynamique du modèle appris et la dynamique du système modélisé. Un autre filtre compare tous les modèles T-AN appris entre eux et élimine ceux qui sont moins probablement corrects. Et nous introduisons enfin un autre filtre qui fait face aux informations incorrectes qui sont apprises à partir des données bruitées.

(C) Analyse formelle des propriétés dynamiques dans les RRB

Ayant comme objectif une meilleure compréhension de la dynamique du système via son modèle et déchiffrer le comportement complexe des systèmes biologiques, nous développons par ailleurs des méthodes qui réalisent une analyse formelle et exhaustive des RRB.

Vérification de la propriété d'atteignabilité :
La première propriété que nous étudions est "l'*atteignabilité*". Elle consiste à répondre à la question suivante : "à partir d'un état global donné du réseau, est-il possible, au cours de l'évolution de la dynamique du réseau d'atteindre un certain objectif ?". Nous notons que dans notre cas, un objectif est un ensemble de niveaux d'expression discrets de différents composants du réseau.

La méthode que nous proposons est une nouvelle approche logique dont l'originalité réside principalement dans le fait qu'elle énumère de façon exhaustive tous les chemins possibles, d'une certaine longueur, tels qu'ils vérifient tous la propriété d'atteignabilité dans un AN et dans un T-AN. Nous avons d'ailleurs développé une variante de notre méthode afin d'identifier le chemin le plus court qui permet d'atteindre l'objectif recherché.

Nous notons que la dynamique des T-AN est calculée selon la sémantique que nous proposons dans cette thèse pour affiner la dynamique des AN après l'intégration des délais dans les interactions. En ce qui concerne la dynamique des AN, nous calculons leurs évolutions selon la sémantique *synchrone* ou *asynchrone*.

Identification des attracteurs :

La deuxième propriété à laquelle nous nous sommes intéressés est l'identification des *attracteurs* dans un AN : un attracteur est un domaine piège minimal, c'est-à-dire une partie du graphe du transition d'états utilisé précédemment dont il n'est pas possible de s'échapper.

De telles structures sont des composants terminaux de la dynamique. Ils peuvent être des singletons, appelés aussi *points fixes*, ou cycliques (non singletons). Nous introduisons deux nouvelles méthodes différentes pour identifier ces deux types d'attracteurs. Elles se sont montrées très efficaces pour faire face à ces problèmes qui sont NP-complets. D'autre part, nous réalisons une étude théorique approfondie sur l'étude des attracteurs cycliques et nous exhibons une nouvelle approche novatrice pour les identifier.

Nous notons que ces vérifications des propriétés dynamiques peuvent aussi entrer dans l'étape de la validation d'un modèle après son inférence (c'est-à-dire après la contribution **(B)**).

Simulation : validation et prédiction de la dynamique du modèle :

Pour les trois phases de contributions citées ci-dessus (**(A)**, **(B)** et **(C)**), nous avons le but de simuler pour *valider* le modèle appris et/ou de simuler pour *prédire* une nouvelle caractéristique comportementale de la dynamique du modèle ou encore parfois de simuler pour vérifier des propriétés dynamiques. Pour les modèles que nous étudions, le résultat de la simulation est une succession d'états du réseau décrivant l'évolution de sa dynamique au cours du temps (e.g., un graphe d'états).

Ainsi, nous avons mis en œuvre un simulateur qui respecte les hypothèses que nous avons adoptées par rapport à la sémantique de la dynamique des T-AN qui modélisent les RRB. Et aussi pour les RRB qui sont modélisés avec le formalisme des AN, nous avons développé un simulateur qui respecte la sémantique purement synchrone et un autre qui respecte la sémantique purement asynchrone. En effet, ces simulateurs nous sont utiles pour l'ensemble des applications et des expériences effectuées dans cette thèse.

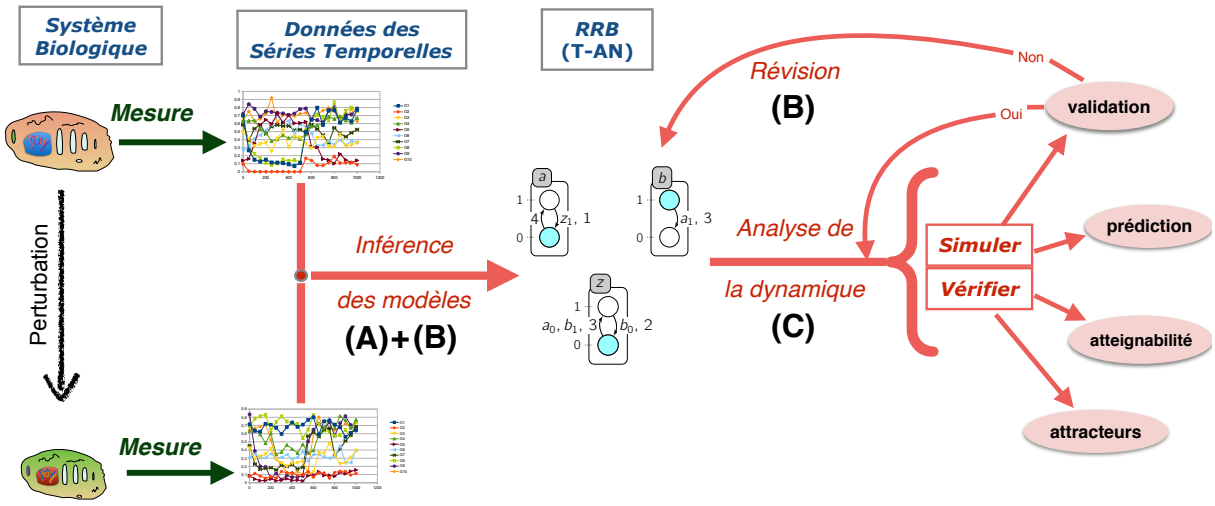


Figure 1.3 : Un schéma global de l'ensemble de la motivation de la thèse. Nous y positionnons les contributions **(A)**, **(B)** et **(C)** de cette thèse.

La plupart des méthodes développées et introduites dans cette thèse (principalement celles des contributions **(B)** et **(C)**) sont mises en œuvre par des implémentations efficaces en utilisant la programmation logique : l'*Answer Set Programming* (ASP) (Baral, 2003). ASP a été identifié dans les années 90 en tant que sous partie de la programmation logique, ayant une sémantique particulière que nous montrons dans le chapitre 6.

L'utilisation d'ASP est considérée comme innovante dans le domaine de l'apprentissage des modèles et de l'analyse des propriétés dynamiques du modèle. En effet, il s'est avéré qu'ASP est particulièrement adapté à la résolution de problèmes combinatoires complexes de recherche (NP-complet) (Baral, 2008). En outre, grâce à l'utilisation d'ASP, nous avons pu développer des approches logiques qui nous permettent de proposer une analyse dynamique qui ne conserve que des informations cohérentes entre elles. En plus, il est très approprié pour la déduction d'informations à partir de connaissances éventuellement non déterminées à condition qu'elles ne soient pas contredites : un aspect non monotone qui est très indiqué pour traiter des données incomplètes dans l'apprentissage.

1.4 Organisation du manuscrit

Le présent manuscrit est organisé de la manière suivante.

Le chapitre 2 propose un état de l'art : (i) sur les principaux formalismes de modélisation des RRB ; (ii) sur les différentes techniques existantes qui permettent l'apprentissage des RRB à partir des données des séries temporelles ; (iii) et enfin sur les principales analyses formelles (statiques et dynamiques) développées pour la vérification des propriétés dynamiques dans les RRB. Nous précisons que nous nous intéresserons principalement aux modèles discrets et leurs raffinements par l'intégration des paramètres temporels.

Le chapitre 3 introduit le formalisme des T-AN qui est une extension des AN après l'intégration du paramètre temporel (i.e., un délai). Nous montrons dans ce chapitre comment cet enrichissement permet de restreindre la dynamique générale des AN pour réussir à avoir un comportement plus réaliste. Nous y développons par ailleurs la sémantique que nous proposons et qui assure ce raffinement de la dynamique. À la fin de ce chapitre

nous comparons le formalisme T-AN par rapport à quelques autres modèles temporels existants.

Le chapitre 4, propose une nouvelle méthode qui infère des RRB modélisés avec le formalisme des T-AN à partir des données de séries temporelles. L'objectif principal de l'approche présentée est de réussir à apprendre un modèle T-AN et qu'il soit aussi cohérent que possible vis-à-vis la dynamique observée du système via les données de séries temporelles à partir desquelles il est appris. Ce travail est effectué en collaboration avec Tony Ribeiro[1] et Katsumi Inoue[2]. Ensuite, nous introduisons les raffinements des modèles T-AN appris qui sont réalisés par l'application d'un ensemble de filtres. Pour montrer l'efficacité de ces filtres, nous comparons les résultats de l'inférence sans et avec les filtres. Les résultats sont obtenus à partir des données synthétiques issues des modèles représentants des systèmes biologiques réels. Puis, nous détaillons dans ce chapitre, la méthode qui révise les T-AN selon des nouvelles données de séries temporelles. Finalement, nous présentons une application de la méthode d'inférence sur des données réelles (fournies par Franck Delaunay[3]) qui portent sur le système de l'horloge circadienne du foie.

Le chapitre 5 se concentre sur l'analyse formelle des propriétés dynamiques des RRB modélisés avec le formalisme des T-AN et des AN. Nous étudions principalement deux propriétés dynamiques : (i) la vérification de l'atteignabilité d'un objectif (un ou plusieurs composants du système) à partir d'un état global du réseau initialement donné, (ii) et l'identification des attracteurs singletons (i.e., points fixes) et non singletons (attracteurs cycliques). Nous rappelons qu'un attracteur est un ensemble d'états globaux dans lesquels le réseau cycle indéfinément sans pouvoir en sortir. Plusieurs particularités et caractéristiques sont déduites à propos de ces propriétés dynamiques et sont démontrés dans ce chapitre. Cet travail est effectué en collaboration avec Maxime Folschette[4].

Le chapitre 6 traite les implémentations en programmation logique : en ASP et la mise en œuvre des approches introduites dans les chapitres qui le précédent. Nous montrons aussi dans ce chapitre l'efficacité de chaque méthode développée via son application sur des données des systèmes biologiques réels.

Le chapitre 7 conclut cette thèse par une discussion récapitulative des différentes contributions apportées et par une ouverture sur des nouvelles perspectives.

1.5 Collaborations

Ce travail de thèse s'inscrit par ailleurs dans le projet de recherche ANR HyClock[5] dont l'intitulé est "Modélisation Hybride Formelle du Temps pour la Biologie des Horloges Circadiennes et la Chronopharmacologie", et qui s'étend d'octobre 2014 à mars 2018. Le présent travail répond notamment à certains objectifs du projet, qui concerne l'apprentissage

[1] Le lien vers sa page personnelle est : `http://www.tonyribeiro.fr/`

[2] Le lien vers sa page personnelle est : `http://research.nii.ac.jp/~inoue/official/content_e.html`

[3] Le lien vers sa page personnelle est : `http://ibv.unice.fr/FR/equipe/delaunay.php`

[4] Le lien vers sa page personnelle est : `http://maxime.folschette.name`

[5] Le site du projet est disponible à : `http://www.agence-nationale-recherche.fr/?Projet=ANR-14-CE09-0011`.

de modèles intégrant la variable du temps afin de représenter le système de l'horloge circadienne.

Une partie du travail de cette thèse a été réalisé dans le cadre d'un stage doctoral dans l'équipe de recherche de Katsumi Inoue, au National Institute of Informatics (NII à Tokyo, Japon). Le sujet de ce stage était : "Learning, Modeling and Reasoning of Dynamic Systems". La sélection de ma candidature a été faite par la direction du programme *Atlanstic* de la région du Pays de la Loire conjointement avec celle à NII. Le laboratoire "Inoue Laboratory" au NII, a participé financièrement à ce travail.

1.6 Notations

Nombres réels :

On note $\mathbb{R}$ l'ensemble des nombres réels. Si $i, j \in \mathbb{R}$, on note $[i\,;j] = \{x \in \mathbb{R} \mid i \leq x \leq j\}$ l'ensemble des nombres réels entre i et j compris.

Entiers naturels :

On note $\mathbb{N}$ l'ensemble des entiers naturels, $\mathbb{N}^* = \mathbb{N} \setminus \{0\}$ l'ensemble des entiers naturels strictement positifs, et $\mathbb{N}^\bullet = \mathbb{N} \setminus \{0, 1\}$ l'ensemble des entiers naturels supérieurs ou égaux à 2.

Si $i, j \in \mathbb{N}$, $i < j$, on note $[\![i\,;j]\!] = \{i, i+1, \ldots, j-1, j\}$ l'ensemble des entiers naturels entre i et j bornes comprises.

Pour tous entiers $i, j, k \in \mathbb{N}$, on note : $k < [\![i\,;j]\!] \overset{def}{\Leftrightarrow} k < i$ et $k > [\![i\,;j]\!] \overset{def}{\Leftrightarrow} k > j$.

Séquences :

Si $n \in \mathbb{N}$, on note $(e_1\,;\ldots;e_n)$ la séquence finie formée des éléments $e_1, \ldots, e_n$ si $n \geq 1$, et on note ε la séquence vide (si $n = 0$).

Pour toute séquence finie $E = (e_1\,;\ldots;e_n)$, on note $|E| = n$ la longueur de cette séquence, et $\mathbb{I}^E = [\![1\,;|E|]\!]$ l'ensemble des indices de cette séquence. Pour tout $i \in \mathbb{I}^E$, on note $E_i = e_i$ le i^{e} élément de E, et pour tout $i, j \in \mathbb{I}^E$, on note $E_{i..j} = (e_i\,;\ldots;e_j)$ la sous-séquence formée des éléments de i à j de E ; naturellement, $E_{i..j} = \varepsilon$ si $i > j$.

On note de plus : $e \in E \Leftrightarrow \exists i \in \mathbb{I}^E, e = E_i$

Ensembles :

Le cardinal d'un ensemble A est noté $|A|$ et son ensemble des parties est noté $\wp(A)$.

Si A et B sont deux ensembles, on note $A \cup B$ leur union, $A \cap B$ leur intersection et $A \times B$ leur produit cartésien.

Si $n \in \mathbb{N}$, et $\{A_i\}_{i \in [\![1;n]\!]}$ est un ensemble d'ensembles, on note $\bigcup_{i \in [\![1;n]\!]} A_i = A_1 \cup A_2 \cup \ldots \cup A_n$ leur union, $\bigcap_{i \in [\![1;n]\!]} A_i = A_1 \cap A_2 \cap \ldots \cap A_n$ leur intersection et $\bigotimes_{i \in [\![1;n]\!]} A_i = A_1 \times A_2 \times \ldots \times A_n$ leur produit cartésien.

Recouvrement :

L'opérateur $\Subset$, qui désigne le recouvrement d'un état par un autre, est donné à la définition 2.11 en page 44.

Chapitre 2

Préliminaires

Nous introduisons dans ce chapitre un tour d'horizon des différentes techniques de modélisation, d'analyse et d'apprentissage des réseaux de régulation biologique (RRB). Nous nous concentrons d'abord sur les abstractions courantes des RRB : le graphe des interactions, les modèles discrets et les modèles hybrides. Nous détaillons aussi les choix qui s'offrent au modélisateur, notamment en matière de sémantique et le raffinement par l'intégration du temps. Ensuite, nous introduisons quelques approches de la littérature qui apprennent les RRB modélisant un système biologique à partir de ses données expérimentales illustrant son évolution dynamique. Finalement, nous présentons aussi le formalisme des frappes de processus, (*Process Hitting*, PH) qui est une sous-classe des réseaux d'automates (*Automata Networks*, AN). Nous montrons alors comment les AN englobent les PH. En effet, le formalisme des AN est la base des travaux présentés dans cette thèse. Il permet de définir des méthodes d'apprentissage et d'analyse efficaces que nous introduirons dans les chapitres suivants de ce manuscrit. Nous faisons alors une rapide revue des principaux travaux qui ont concerné ce formalisme. En outre, nous avons mis en évidence ses avantages pour traiter des modèles de grande taille.

2.1 Introduction

Les premiers formalismes de modélisation utilisés pour étudier les RRB ont été basés sur les systèmes d'*équations différentielles ordinaires* (EDO) qui ont été proposées pour capturer le comportement du système étudié, par exemple dans (Mesarović, 1968). Dans les EDO, la modélisation est *quantitative* ; chaque variable du modèle mathématique abstrait la concentration du produit d'un composant biologique particulier (e.g., un gène), et les changements de concentration s'expliquent par la différence entre le taux de synthèse et le taux de dégradation. Toutefois, biologiquement, il est possible de considérer que

les régulations de gènes sont *qualitatives* (chaque régulation est alors distinguée par son activité ou inactivité). Ce besoin de simplification est justifié par le fait que la valeur d'un grand nombre de paramètres reste inconnue. Dans un certain cas, la modélisation peut être *sem-qualitative* et le modèle mathématique peut devenir un système d'*équations différentielles linéaires par morceaux* : tant que les régulations ciblant un gène ne changent pas d'état (actif ou inactif), le taux de synthèse de ce gène est considéré constant. Plusieurs travaux sont basés sur des équations différentielles linéaires par morceaux, comme dans (De Jong, Gouzé, Hernandez, Page, Sari & Geiselmann, 2004). Si les paramètres cinétiques sont correctement déterminés, ces modèles produisent des dynamiques cohérentes avec des observations expérimentales.

L'inconvénient d'un formalisme si expressif est sa sensibilité aux paramètres cinétiques. De plus, le manque de connaissances quantitatives sur ces paramètres, la difficulté de les mesurer dans le système réel aussi bien que la fiabilité des valeurs obtenues, obligent souvent à utiliser pour les simulations des valeurs approximatives si ce n'est plus ou moins arbitraires. En outre, ces simulations produisent les trajectoires de la dynamique du système, mais ne permettent pas souvent de conclure sur l'effet d'une action particulière sur le système. Plus précisément, les simulations faites avec des valeurs approximatives de paramètres cinétiques ne permettent pas réellement de déduire toutes les conséquences d'une infime modification du système.

Pour faire face à ces limitations, certains chercheurs préfèrent travailler avec des modèles *qualitatifs* en considérant des domaines où les variables ont des taux de synthèses quasi constants afin de faire du *model-checking* sur les systèmes modélisés par équations différentielles par morceaux.

Nous ne détaillons pas dans ce chapitre un état de l'art sur les modélisations continues (en l'occurrence sur les ODE), car nos travaux sont plutôt basés sur la modélisation discrète qualitative. Une étude a été faite dans (De Jong, 2002) pour montrer le lien entre ces deux approches de modélisation.

La section 2.2 présente trois niveaux de spécification des RRB couramment utilisés, correspondant à différents degrés d'abstraction du système étudié :

- Le *graphe des interactions* peut être considéré comme la spécification la plus abstraite d'un RRB : seules les informations qualitatives de régulation entre les composants sont référencées (a est un activateur de b, par exemple).
- Les *modélisations discrètes* associent à chaque composant un ensemble fini discret de niveaux qualitatifs ({0, 1, 2}, par exemple), reflétant une quantification de sa concentration réelle, et caractérisent l'évolution des composants en fonction de leurs régulateurs. Cette dernière peut par exemple être définie par les paramètres discrets de René Thomas (Thomas, 1991).
- Enfin, les *modélisations hybrides* ajoutent une dimension continue gouvernant les transitions discrètes du système, généralement par l'attribution de délais (aléatoires ou par intervalle de temps) à l'application des régulations.

Ensuite, la section 2.3 dresse un état de l'art des analyses statiques et dynamiques opérant sur les RRB. Deux propriétés dynamiques sur lesquelles nous nous concentrons :

- la vérification de l'atteignabilité, et
- l'identification des points fixes et des attracteurs cycliques.

Puis, la section 2.4 introduit quelques méthodes de la littérature dont le but est d'apprendre des RRB à partir de leurs dynamiques observées lors des expérimentations. Nous détaillons ainsi dans cette section quelques avantages et quelques inconvénients de ces méthodes.

Finalement, après un préambule sur les méthodes de modélisation et d'analyse des RRB, la section 2.5 donne la représentation des RRB avec les frappes de processus (*Process Hitting*, PH), est un formalisme introduit dans la thèse de Loïc Paulevé (Paulevé, 2011). Ensuite, ce formalisme a subi plusieurs enrichissements au cours des dernières années et c'est ce qui nous ramène, dans cette thèse, à travailler avec les réseaux d'automates (*Automata Networks*, AN). En effet, les PH forment une sous classe des AN. Nous défendons ainsi, à la fin de cette section 2.5 le choix de ce formalise.

2.2 Modélisation des RRB

Cette section détaille les principales méthodes de modélisation des RRB. Ces méthodes englobent des modélisations discrètes et des modélisations hybrides. Nous commençons par présenter le graphe des interactions, qui permet d'introduire, par la suite, les modèles discrets (comme les modèles de Kaufmann et de Thomas). Puis nous introduisons d'autres modélisations hybrides utilisées pour la modélisation des RRB.

2.2.1 Graphes des interactions

Le *graphe des interactions* d'un RRB présente les régulations entre les composants avec des réseaux simples. En effet, les composants biologiques sont représentés par des nœuds étiquetés par un nom (celui du composant : a, b, c, etc.) et les interactions par des arcs orientés signés (positifs ou négatifs). Un arc signé positif (resp. négatif) de a vers b dénote que a est un activateur (resp. inhibiteur) de b.

La figure 2.1 (gauche) ci-dessous présente un graphe des interactions simple tel qu'un arc positif (resp. négatif) est étiqueté par le signe $+$ (resp. $-$).

Afin de mieux identifier le rôle des régulations impliquées, le graphe des interactions peut être agrémenté d'informations supplémentaires en se basant sur le niveau de connaissance du système et les questions posées. Nous illustrons dans la figure 2.1, ces différents niveaux de spécification d'un RRB et nous les expliquons dans la suite.

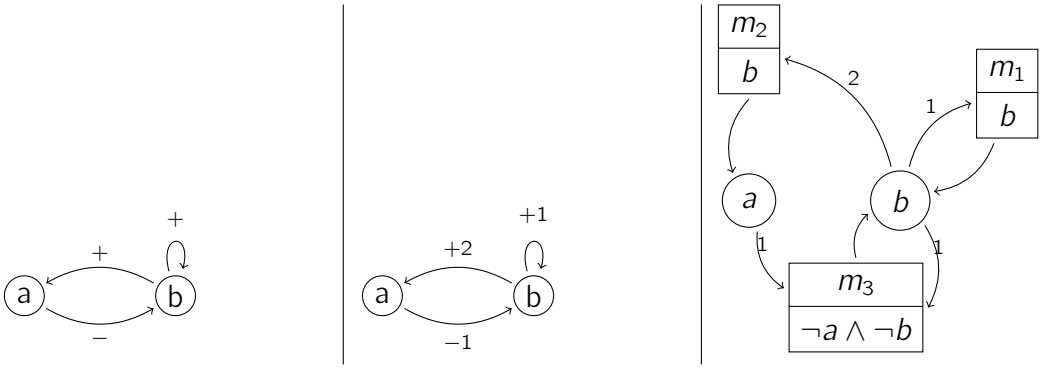


Figure 2.1 : Différents niveaux de représentation du graphe des interactions d'un RRB : (gauche) graphe simple ; (milieu) graphe avec seuils ; (droite) graphe avec multiplexes.

Il est possible d'ajouter aux nœuds d'un graphe des interactions un plafond (son niveau d'expression maximum : l_a, pour le composant a) et les arcs peuvent prendre la forme $a \xrightarrow{s,l} b$, c'est-à-dire étiquetés par un signe s qui représente le type de la régulation ($+$ pour une activation, $-$ pour une inhibition et $\circ$ pour une régulation plus complexe) et un entier l qui représente le seuil de déclenchement de la réaction (c'est-à-dire le niveau d'expression du composant régulateur à partir duquel celui-ci a effectivement une influence sur le composant régulé). Lorsque ce seuil n'est pas atteint, la régulation peut être considérée soit inoffensive, soit inversée (b devient un inhibiteur de a). La notion de seuil (niveau) est présentée plus en détail dans la suite dans les réseaux discrets. Un exemple d'un *graphe des interactions avec les seuils* est montré dans la figure 2.1 (milieu). Par exemple, l'arc de b vers a étiqueté $+2$ représente la régulation $b \xrightarrow{+,2} a$. En d'autres termes, b se comporte comme un activateur de a si son niveau d'expression est supérieur ou égal à la valeur arbitraire 2, sinon il se comporte comme un inhibiteur (c'est-à-dire si son niveau d'expression est égal à 1 ou à 0).

Étudions avec plus de détails les modèles RRB : il s'avère que plusieurs régulations s'effectuent en coopération entre des différents composants. Par exemple si a et b sont des activateurs de c, dans certains cas, l'activation de c n'est possible que si a et b sont à la fois présents. Ce type de coopération (comme celle entre a et b) correspond dans certains systèmes biologiques à la formation de complexes. Les coopérations peuvent aussi s'effectuer entre les composants présents et les composants absents. Autrement dit, selon les seuils des composants le complexe peut être formé ou pas. Nous notons que de telles coopérations peuvent être spécifiées dans le graphe des interactions à travers l'introduction des multiplexes (Bernot, Comet & Khalis, 2008). L'enrichissement des graphes des interactions par des multiplexes permet (1) une appréhension plus facile du modèle à la simple lecture du graphe des interactions et (2) un raffinement des dynamiques spécifiées et donc une réduction du champ de recherche des paramètres discrets permettant de spécifier complètement le comportement du système.

La figure 2.1(droite) montre un exemple de graphe des interactions avec multiplexes qui sont représentés par des rectangles divisés en deux parties : la partie haute contient le nom du multiplexe et la partie basse spécifie l'expression booléenne logique de son activation. Une expression booléenne logique contient les variables correspondant aux composants formant le multiplexe : a est vrai si et seulement si le niveau du composant a est supérieur ou égal au seuil de l'arc entrant sur le multiplexe. Ainsi, dans le cas de m_3, tel que $m_3 = \neg a \wedge \neg b$ est vrai si et seulement si le niveau de a est inférieur à 1 et aussi le niveau de b inférieur à 1 (c'est-à-dire 0).

La définition 2.1 propose une formalisation générale du *graphe des interactions*. Nous notons que plusieurs définitions dans cette section sont prises de (Folschette, 2014).

Définition 2.1 (Graphe des interactions)**.** Un *graphe des interactions* est un couple $\mathcal{G} = (N\,;E)$ où N est l'ensemble fini des *composants*, étiquetés par un nom et un *plafond*, et E est l'ensemble fini des *régulations* entre deux nœuds, étiquetées par un *signe* et un *seuil* :

$$E \stackrel{def}{=} \{a \xrightarrow{s,l} b, \ldots \mid a, b \in N \wedge s \in \{+,-,\circ\} \wedge l \in [\![1\,;l_a]\!]\}$$

tel que chaque régulation de a vers b soit unique :

$$\forall a \xrightarrow{s,l} b \in E, \forall a \xrightarrow{s',l'} b \in E, s = s' \wedge l = l' \ .$$

Étant donnée cette définition, on note $E_s \stackrel{def}{=} \{a \xrightarrow{s,l} b \in E\}$ pour $s \in \{+, -, \circ\}$. De plus, pour tout composant $b \in N$, on note $\mathcal{G}^{-1}(b)$ l'ensemble de ses *régulateurs* :

$$\mathcal{G}^{-1}(b) \stackrel{def}{=} \{a \in N \mid a \xrightarrow{s,l} b \in E\}$$

De nombreuses bases de données représentent les RRB sous forme de graphes des interactions. Nous donnons quelques exemples de ces bases de données qui regroupent les RRB sous forme de graphes des interactions : Pathway Interaction Database (PID) (Schaefer, Anthony, Krupa, Buchoff, Day, Hannay & Buetow, 2009), Transcriptional Regulatory Relationships Unraveled entence-based Text mining (TRRUST) (Han, Shim, Shin, Shim, Ko, Shin, Kim, Cho, Kim, Lee et al., 2015), Kyoto Encyclopedia of Genes and Genomes (KEGG) (Kanehisa, Sato, Kawashima, Furumichi & Tanabe, 2015), Reactome (Joshi-Tope, Gillespie, Vastrik, D'Eustachio, Schmidt, de Bono, Jassal, Gopinath, Wu, Matthews et al., 2005) et WikiPathways (Pico, Kelder, Van Iersel, Hanspers, Conklin & Evelo, 2008) etc. Ainsi plusieurs formats en dérivent (par exemple, ZGINML, SMBL, SIF, BIOPAX, SBGN, KGML, etc.). Dans la plupart des cas, ces formats sont interopérables pour permettre un plus grand partage des modèles.

Ces bases de données contiennent les informations provenant de la littérature, des résultats obtenus des publications basées sur des recherches expérimentales. Ce sont des réseaux de protéines ou de signalisations très utiles pour l'étude des propriétés topologiques des réseaux (Ma'ayan, Jenkins, Neves, Hasseldine, Grace, Dubin-Thaler, Eungdamrong, Weng, Ram, Rice et al., 2005) ou d'un mappage de données (Ideker & Sharan, 2008; Terfve & Saez-Rodriguez, 2012).

En revanche, dans la plupart des cas, ces bases de données ne contiennent pas d'information sur les seuils des interactions entre les composants. Dans la suite, nous allons nous intéresser aux modélisations discrètes.

2.2.2 Les modèles discrets

Définition des réseaux booléens et des réseaux discrets

Dans les modélisations discrètes, chaque composant a un ensemble fini et dénombrable de niveaux qualitatifs. Ces niveaux ne sont qu'une abstraction de la dynamique des composants. En effet, chaque niveau représente en général un intervalle précis de la concentration d'un composant. Ces réseaux ont été introduits dans un premier temps par Stuart A. Kauffman (1969) puis par René Thomas (1973) dans le cadre de formalismes booléens qui a ensuite été étendu au multivalué.

En général, l'activation (ou l'inhibition) effective d'un composant a dépend de son niveau d'expression (i.e., niveau de concentration) ainsi que des niveaux de ces régulateurs ($\mathcal{G}^{-1}(a)$). En effet, tant qu'un certain seuil n'est pas atteint, la régulation peut être considérée comme ineffective ou inversée. On veut dire par régulation inversée que l'activateur devient inhibiteur et/ou l'inhibiteur devient activateur jusqu'à ce que le seuil soit atteint.

Ainsi, pour tout composant a régulant b, c'est-à-dire si $a \xrightarrow{s,l} b \in E$, on note $\mathsf{niveaux}(a \to b)$ (resp. $\overline{\mathsf{niveaux}}(a \to b)$) l'ensemble des niveaux d'expression de a qui sont au-dessus (resp. en-dessous) du seuil l (voir définition 2.2).

Exemple. La figure 2.1 en page 25 (milieu) ci-dessus, représente un graphe des interactions $(N\,;E)$ où $N=\{a,b\}$, avec $l_a=1$ et $l_b=2$ les niveaux des seuils maximums de a et b respectivement, et :

$$E_+=\{b \xrightarrow{+,2} a,\ b \xrightarrow{+,1} b\} \qquad E_\circ=\emptyset \qquad E_-=\{a \xrightarrow{-,1} b\}$$

Ainsi :

$$\mathcal{G}^{-1}(a)=\{b\} \qquad \mathcal{G}^{-1}(b)=\{a,b\}$$

Définition 2.2 (Niveaux effectifs (niveaux))**.** Soit $\mathcal{G}=(N\,;E)$ un graphe des interactions tel que défini à la définition 2.1 en page 26.
Si $a \xrightarrow{s,l} b \in E$, on définit :

$$\mathsf{niveaux}(a\to b) \overset{def}{=} [\![l\,;l_a]\!] \quad \text{et} \quad \overline{\mathsf{niveaux}}(a\to b) \overset{def}{=} [\![0\,;l-1]\!]$$

Exemple. Sur le graphe des interactions de la figure 2.2 (gauche) on a notamment :

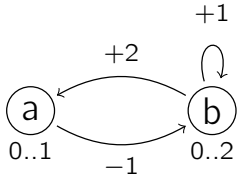
$$\mathsf{niveaux}(a\to b)=[\![1\,;1]\!] \qquad \overline{\mathsf{niveaux}}(a\to b)=[\![0\,;0]\!]$$

Le paramètre de René Thomas $K_{a,\omega}$ introduit le niveau vers lequel évolue le composant a soumis aux interactions positives et négatives des composants dans ω. En effet, un paramètre $K_{a,\omega}$ représente un ensemble de valeurs vers lesquelles le composant a évolue dans tout état où l'ensemble de ses ressources est égal à ω. Autrement dit, a va évoluer vers la valeur de $K_{a,\omega}$ qui est la plus proche de son niveau d'expression courant. Nous présentons dans la définition 2.3 une formalisation du paramètre $K_{a,\omega}$.

Définition 2.3 (Paramètre $K_{a,\omega}$ et paramétrisation K)**.** Soit $\mathcal{G}=(N\,;E)$ un graphe des interactions. Pour un composant $a\in N$ donné et $\omega\subset\mathcal{G}^{-1}(a)$ un sous-ensemble de ses régulateurs, le *paramètre* $K_{a,\omega}=[\![i\,;j]\!]$ est un intervalle non-vide tel que $0\le i\le j\le l_a$. La carte complète K des paramètres sur un graphe des interactions $\mathcal{G}$ est appelée *paramétrisation de* $\mathcal{G}$.

On en déduit alors qu'un modèle de *Thomas* est un couple formé d'un graphe des interactions et d'une paramétrisation et noté $(\mathcal{G}\,;K)$.

Exemple. Considérant la paramétrisation suivante pour le graphe des interactions de la figure 2.1 en page 25.



$$K_{a,\emptyset}=[\![0\,;0]\!] \qquad K_{b,\emptyset}=[\![1\,;2]\!]$$
$$K_{a,\{b\}}=[\![0\,;1]\!] \qquad K_{b,\{a\}}=[\![0\,;0]\!]$$

Figure 2.2 : (gauche) Un exemple de graphe des interactions avec seuils (i.e., un réseau discret). (droite) Un exemple de paramétrisation du graphe des interactions de gauche. Nous définissons les paramètres $K_{a,\omega}$ à la définition 2.3 à la page 28.

Concernant la dynamique du modèle, pour tout niveau d'expression de a appartenant à $\text{niveaux}(a \to b)$, a est censé avoir une influence, correspondante au signe s, sur b ; c'est-à-dire être activateur si $s = +$, inhibiteur si $s = -$, ou avoir une influence indéterminée ou multiple si $s = \circ$. En revanche, pour tout niveau d'expression de a appartenant à $\overline{\text{niveaux}}(a \to b)$, l'influence opposée devrait être effective.

Cette hypothèse permet de modéliser la dégradation de b en l'absence de l'activation de a si $s = +$, ou l'activation de b en l'absence de l'inhibition de a si $s = -$.

Les réseaux discrets sont fondés sur un graphe des interactions, mais ils peuvent utiliser des fonctions d'évolution (définition 2.4) pour plus de permissivité, à la place des paramètres discrets tels que précédemment formalisés à la définition 2.3 ci-dessus. En effet, chaque fonction, qui est spécifique pour un composant (f_a pour a), calcule le nouveau niveau discret vers lequel le composant a évolue. Ceci dépend alors principalement du niveau de a et des niveaux de ses régulateurs.

Définition 2.4 (Réseau discret (RD))**.** Si $\mathcal{G} = (N ; E)$ est un graphe des interactions, un *réseau discret* est un couple $\text{RD} = (\mathcal{G} ; F)$ avec $F = (f_a)_{a \in N}$, tels que
$\forall a \in N, f_a : \mathcal{G}^{-1}(a) \to [\![0 ; l_a]\!]$.

Est désigné par *réseau booléen* tout réseau discret dont les composants possèdent au plus deux niveaux discrets, c'est-à-dire : $\forall x \in N, l_x = 1$.

Pour modéliser les RRB, des modèles avec différentes sémantiques de mise à jour de la dynamique ont été proposés, à savoir le *synchrone* (Kauffman, 1969) l'*asynchrone* (Thomas, 1991) ou encore des modèles combinant entre le synchrone et l'asynchrone (Fauré, Naldi, Chaouiya & Thieffry, 2006). Il est clair qu'il est important d'utiliser une sémantique de mise à jour adéquate pour tendre à obtenir une dynamique du modèle telle qu'elle soit biologiquement plus correcte (c'est-à-dire plus proche de la dynamique du système réel qu'elle modélise). En effet, les dynamiques synchrone et asynchrone des modèles des RRB ont été abordées de façon qualitative dans le passé (Garg, Di Cara, Xenarios, Mendoza & De Micheli, 2008; Noual & Sené, 2017). Et des études ont montré que différentes sémantiques de la dynamique peuvent conduire à différents attracteurs (plus de détails dans le chapitre 5).

Dans un modèle dynamique synchrone, à chaque instant, plusieurs gènes modifient leurs niveaux d'expression simultanément. Ce type de sémantique est convenable pour les réseaux très volumineux. En revanche, elle est au détriment de la précision car, en réalité, des gènes différents prennent un temps différent pour passer d'un niveau d'expression à un autre. Tel est le cas des modèles dynamiques asynchrones, dans lesquels tous les gènes prennent des temps différents pour faire une transition d'un niveau vers un autre. Par contre, ils ont une dynamique qui est plus complexe à modéliser et à analyser.

Dans la suite, et dans les deux sous sections suivantes, nous donnons plus de détails sur ces deux sémantiques de la dynamique : asynchrones et synchrones. Et nous introduisons la dynamique des réseaux discrets asynchrone (RDA) puis la dynamique des réseaux discrets synchrone (RDS).

La dynamique des réseaux discrets asynchrones

La sémantique asynchrone a été principalement défendue par René Thomas en (1991). La dynamique d'un modèle de *Thomas* $(\mathcal{G} ; K)$ suit deux hypothèses formulées par René

Thomas lui-même.

- **Asynchrone** : un seul composant peut évoluer entre chaque état. En effet, il est vraiment très peu probable que deux composants passent en même temps un seuil d'expression discret.
- **Unitaire** : chaque composant ne peut évoluer que d'un niveau discret à la fois.

Dans la définition 2.5 ci-dessous, l'hypothèse d'asynchronisme est conservée car un seul composant peut évoluer depuis chaque état donné.

Définition 2.5 (Dynamique d'un réseau discret asynchrone ($\rightarrow_{\mathsf{RDA}}$))**.** Pour tout réseau discret asynchrone $\mathsf{RDA} = (\mathcal{G}\,;F)$, la dynamique unitaire de RDA est donnée par la relation de transition $\rightarrow_{\mathsf{RDA}} \in \mathcal{S}\times\mathcal{S}$ définie par :

$$\begin{aligned}\forall s,s'\in\mathcal{S}, s\rightarrow_{\mathsf{RDA}} s' \iff \exists a\in N, s[a]\neq f_a(s)\wedge s'[a]=f_a(s)\\ \wedge\,\forall b\in N, b\neq a\Rightarrow s[b]=s'[b]\end{aligned}$$

Il y a certaines contraintes spécifiques aux modèles de Thomas qui peuvent être relâchées afin qu'on puisse observer dans le système des comportements supplémentaires. Ce qui justifie la représentation courante de la dynamique des RRB par des *réseaux discrets*. En effet, pour la dynamique du modèle de *Thomas* avec des paramètres discrets est définie comme suit : il existe une transition d'un état s vers un état s' si et seulement si il existe un unique composant a qui évolue entre ces deux états, d'exactement un niveau d'expression et vers le paramètre $K_{a,\mathsf{Res}_a(s)}$ (voir définition 2.7 ci-dessous). Il est à noter que a ne peut évoluer que si son niveau d'expression dans l'état s appartient déjà à l'intervalle du paramètre $K_{a,\mathsf{Res}_a(s)}$.

Définition 2.6 (Ressources (Res))**.** Soit $\mathcal{G}=(N\,;E)$ un graphe des interactions. Pour tout composant $a\in N$ et tout état s du graphe, on appelle *ressources de a dans s* et on note $\mathsf{Res}_a(s)$ l'ensemble des régulateurs de a dont le niveau dans s est supérieur au seuil de la régulation qui les relie à a :

$$\mathsf{Res}_a(s)\stackrel{def}{=}\{b\in\mathcal{G}^{-1}(a)\mid s[b]\in\mathsf{niveaux}(b\rightarrow a)\}$$

avec $s[b]$ est le niveau de b dans s.

Définition 2.7 (Dynamique unitaire d'un modèle de Thomas ($\rightarrow$))**.** Pour tout modèle de Thomas $\mathsf{T}=(\mathcal{G}\,;K)$, tel que $\mathcal{G}=(N\,;E)$ le graphe des interactions et K les paramètres de Thomas de ce graphe ; la dynamique de T est donnée par la relation de transitions $\rightarrow\subseteq\mathcal{S}\times\mathcal{S}$ définie par :

$$\begin{aligned}\forall s,s'\in\mathcal{S}, s\rightarrow s' \iff \exists a\in N, s[a]\notin K_{a,\mathsf{Res}_a(s)}\wedge s'[a]=s[a]+\delta^a(s)\\ \wedge\,\forall b\in N, b\neq a\Rightarrow s[b]=s'[b]\end{aligned}$$

$$\text{avec : } \delta^a(s)=\begin{cases}+1 & \text{si } s[a]<K_{a,\mathsf{Res}_a(s)}\\ -1 & \text{si } s[a]>K_{a,\mathsf{Res}_a(s)}\end{cases}$$

Les symboles « $<$ » et « $>$ » de cette définition permettant de comparer un entier à un intervalle sont définis à la section 1.6 en page 22.

Exemple. Reprenons la figure 2.2 en page 28 qui montre un exemple de paramétrisation du graphe des interactions de la figure 2.2 (gauche). Elle présente un modèle de Thomas complet, dont la figure 2.3 ci-dessous donne l'espace d'états qui représente une évolution du modèle. Les états y sont représentés par des couples $\langle a_i, b_j \rangle$ où i et j représentent respectivement les niveaux d'expression de a et b. Pour ce modèle de Thomas, les transitions entre les états présentées dans le graphe d'états ci-dessous sont calculées d'après la définition 2.7 :

$$\langle a_0, b_2 \rangle \to \langle a_1, b_2 \rangle \to \langle a_1, b_1 \rangle \to \langle a_1, b_0 \rangle$$

avec a_i qui représente le composant a au niveau d'expression discret i. On note notamment la présence d'un état stable (i.e., point fixe) pour ce modèle ; c'est-à-dire un état depuis lequel plus aucune évolution est possible : $\langle a_1, b_0 \rangle$.

On peut observer l'aspect unitaire de la dynamique d'un modèle de *Thomas* sur ce graphe. En effet, malgré la valeur du paramètre $K_{b,\emptyset} = [\![2\,;2]\!]$, le composant b ne peut pas directement passer de l'état b_2 à l'état b_0 en « sautant » l'état b_1. C'est pourquoi on observe les transitions $\langle a_1, b_2 \rangle \to \langle a_1, b_1 \rangle$ et puis $\langle a_1, b_1 \rangle \to \langle a_1, b_0 \rangle$.

$$\begin{array}{ccccc} \langle a_0, b_0 \rangle & \longrightarrow & \langle a_0, b_1 \rangle & \longrightarrow & \langle a_0, b_2 \rangle \\ & & & & \downarrow \\ \langle a_1, b_0 \rangle & \longleftarrow & \langle a_1, b_1 \rangle & \longleftarrow & \langle a_1, b_2 \rangle \end{array}$$

Figure 2.3 : Représentation de la dynamique du modèle de Thomas donné à la figure 2.2. Chaque état est représenté par un couple $\langle a_i, b_j \rangle$ où i et j représentent respectivement le niveau d'expression de a et b et une transition entre deux états est représentée par une flèche.

Dynamique des réseaux discrets synchrones

Dans un modèle parfaitement synchrone (Kauffman, 1969), toutes les fonctions d'évolution sont appliquées en même temps (*simultanément* : tous les niveaux des variables peuvent changer par une transition d'état. D'où le fait que la dynamique du réseau est déterministe, il n'y a qu'une seule évolution possible à partir de chaque état du réseau. Autrement dit, pour chaque état n'a qu'un seul successeur. Nous introduisons formellement la dynamique synchrone des RD dans la définition 2.8.

Définition 2.8 (Dynamique d'un réseau discret synchrone ($\to_{\mathsf{RDS}}$))**.** Pour tout réseau discret synchrone $\mathsf{RDS} = (\mathcal{G}\,;F)$, la dynamique non-unitaire de RDS est donnée par la relation de transition $\to_{\mathsf{RDS}} \subseteq \mathcal{S} \times \mathcal{S}$ définie par :

$$\forall s, s' \in \mathcal{S}, s \to_{\mathsf{RDS}} s' \iff \forall a \in N, s'[a] = f_a(s)$$

2.2.3 Modèles hybrides

Il s'est avéré qu'un enrichissement des modèles est possible par l'ajout d'une composante supplémentaire qui gouverne les transitions entre les états discrets. Ainsi, il a été possible

d'explorer des propriétés quantitatives telles que la probabilité d'observer un comportement et le temps moyen d'exécution etc.

Vu que les dynamiques des RRB sont des évolutions intrinsèquement aléatoires, et que la notion de délais peut jouer un rôle dans la caractérisation d'un comportement donné, nous remarquons une tendance à l'étude vers des modèles hybrides afin de mieux comprendre les systèmes biologiques. Nous présentons dans la suite brièvement un état de l'art de deux types de modélisations hybrides : des modèles enrichis par une composante stochastique et des modèles raffinés par une composante temporelle.

2.2.3.1 Modèles stochastiques

Les modèles stochastiques associent une composante temporelle aléatoire aux transitions qui en général, suit une distribution exponentielle, permettant d'avoir un système markovien. Par conséquent, plusieurs travaux ont été initiés dans le cadre des modélisations markoviennes afin de modéliser et d'analyser des systèmes biologiques. Nous pouvons citer l'utilisation de Réseaux de Petri stochastiques (Heiner, Gilbert & Donaldson, 2008), du π-calcul stochastique (Priami, 1995; Maurin, Magnin & Roux, 2009), de κ (Danos, Feret, Fontana & Krivine, 2007) ou encore de Biocham (Rizk, Batt, Fages & Soliman, 2008).

Le but principal des modélisations stochastiques est de permettre le calcul des probabilités d'observation de certains comportements comme dans (Baier, Bertrand, Bouyer, Brihaye & Größer, 2007). Ainsi, il est possible de prédire les évolutions dynamiques les plus probables par rapport à d'autres. Par ailleurs, dans le cas de la vérification de l'atteignabilité, il serait possible d'avoir une idée sur la probabilité d'atteindre un certain objectif fixé.

2.2.3.2 Modèles temporels

Le but des modélisations temporelles est d'offrir un raffinement sur les dynamiques discrètes initiales. En effet, plusieurs possibilités émergent du fait de la modélisation temporelle : (1) du fait de la contrainte temporelle, il peut être désormais possible d'observer des comportements initialement interdits dans la dynamique discrète et de même, la contrainte temporelle peut aussi interdire certaines transitions ; (2) la composante temporelle permet d'avoir des comportements plus précis sur la dynamique.

Au cours de la dernière décennie, la modélisation hybride des RRB a suscité un intérêt croissant pour les *délais* entre les états du réseau. Ces approches hybrides considèrent différents cadres de modélisation. En général, la composante temporelle dans les modèles temporels représente le délai d'une transition pris dans un intervalle de temps fixé ou bien sa valeur suit une certaine équation différentielle.

Les mérites des formalismes hybrides dans la biologie ont été étudiés, par exemple dans ; les Réseaux de Petri temporisés (Popova-Zeugmann, Heiner & Koch, 2005), les automates temporisés (Siebert & Bockmayr, 2006; Batt, Salah & Maler, 2007), les automates hybrides linéaires (Ahmad, Roux, Bernot, Comet & Richard, 2008; Behaegel, Comet, Bernot, Cornillon & Delaunay, 2016), les automates hybrides non linéaires (Alur, Belta, Kumar, Mintz, Pappas, Rubin & Schug, 2002), le modèle hybride d'un oscillateur neuronal (Mandon, Haar & Paulevé, 2012), et la représentation booléenne (Paoletti, Yordanov, Hamadi, Wintersteiger & Kugler, 2014; Li, Yang & Chu, 2012).

Dans (Matsuno, Doi, Nagasaki & Miyano, 2000), les auteurs passent des réseaux de Petri aux réseaux de Petri hybrides : l'avantage de l'approche hybride en ce qui concerne la

modélisation discrète réside dans la possibilité de capturer des facteurs biologiques, par exemple le délai mis pour la transcription de l'ARN polymérase.

D'autres travaux d'étude de RRB ont aussi été basés sur les équations différentielles avec délais comme dans (Hale & Lunel, 2013). Ces modèles permettent de ne pas spécifier tous les processus explicitement et leurs effets sur la dynamique du système peuvent être regroupés et définis sous la forme de délais. Cette caractéristique est intéressante car des données expérimentales pour le système d'intérêt sont souvent manquantes, et certains des processus qui ne sont pas bien connus ou compris peuvent être regroupés sous forme de délais pour réduire ainsi le nombre de variables et de paramètres du modèle. En revanche, ces équations différentielles avec délais sont difficiles à résoudre et les méthodes géométriques et de plan de phase usuelles ne peuvent pas être utilisées simplement.

Dans (Batt et al., 2007), les auteurs utilisent le formalisme de (Maler & Pnueli, 1995) pour modéliser un RRB. Dans ce formalisme, une distinction est possible entre les gènes et leurs produits. Les gènes sont représentés comme des fonctions booléennes de produits de gènes. Les produits de gène sont représentés par leurs niveaux de concentration et leur évolution (positive ou négative) est en fonction de leur gène (actif ou non). En plus l'action du gène sur le niveau de concentration de son produit est temporisée sur un intervalle de délais. Tout comme dans les travaux de (Siebert & Bockmayr, 2006), ils considèrent l'intervalle de délais pour passer d'un niveau n à un niveau $n \pm 1$ pour une variable donnée. En fait, ces deux approches de modélisation sont très proches bien que le formalisme pour la description du réseau de régulation génétique pour les modéliser soit différent et que Batt et al. (2007) considèrent que les produits de gène évoluent continuellement (leur vitesse d'évolution n'est donc jamais nulle).

Enfin, dans (Comet, Fromentin, Bernot & Roux, 2010), les auteurs étudient une extension directe de l'approche de modélisation discrète de René Thomas en introduisant des délais quantitatifs. Ces délais représentent le temps obligatoire pour qu'un gène passe d'un niveau qualitatif discret au suivant (ou précédent). Ils présentent l'avantage d'un tel cadre pour l'analyse de la production de mucus dans la bactérie *Pseudomonas aeruginosa*. L'approche que nous proposons dans cette thèse dans le chapitre 3 hérite de cette idée que certains modèles doivent saisir ces caractéristiques de synchronisation.

La section 2.5 en page 42 présente le formalisme des Frappes de Processus qui est un modèle discret et une restriction du formalisme des réseaux d'automates. En effet, dans cette thèse, nous utilisons ce dernier pour la modélisation des RRB. Il présente l'avantage de permettre une modélisation hybride des systèmes. Ceci par une modélisation des composants comme des composants à états discrets (Paulevé, 2011; Folschette, 2014) et une dynamique continue qui permet à la fois la prise en compte du temps sous forme continue avec un comportement aléatoire (Fippo-Fitime, 2016).

2.3 Analyse formelle des propriétés dynamiques des RRB

La majorité des systèmes biologiques ont une caractéristique qui les différencie des autres champs d'application de l'informatique. Cette caractéristique consiste au regroupement des entités, ayant un comportement simple, et la production d'un comportement complexe de par leurs interactions. Ceci est alors à l'origine de l'explosion combinatoire des comportements à analyser par les méthodes « classiques ». Afin de contourner cette explosion combinatoire,

plusieurs recherches ont été menées durant la dernière décennie. Par exemple, l'analyse dite *statique* qui réussit à conclure sur des propriétés de la dynamique du modèle étudié sans avoir besoin de l'analyser de façon exhaustive. En effet, les approches d'analyses statiques reposent plutôt sur une analyse des structures du modèle tout en ayant parfois un recours à des méthodes d'abstraction afin d'en simplifier le comportement, comme dans (Feret, Henzinger, Koeppl & Petrov, 2012; Paulevé, Chancellor, Folschette, Magnin & Roux, 2014; Folschette et al., 2015).

Dans cette thèse, les propriétés dynamiques étudiées sont : la vérification de l'*atteignabilité* et l'identification des *attracteurs*. Ainsi, nous présentons dans cette section un état de l'art sur quelques travaux développés pour l'analyse et la vérification de ces deux propriétés.

2.3.1 Atteignabilité

Quand nous vérifions la propriété de l'atteignabilité dans un RRB, c'est-à-dire que nous vérifions l'existence d'un chemin qui permet d'atteindre l'objectif fixé à partir d'un état initial du réseau. Nous présentons dans la suite quelques méthodes existantes pour l'analyse formelle de cette propriété.

Analyse avec abstraction du modèle

Le but principal de la vérification des modèles par abstraction est de produire des analyses efficaces du système sans l'exécuter (Cousot & Cousot, 1977). Plusieurs travaux sont faits en se basant sur une telle approche, dans le domaine de la biologie des systèmes ou autres. Par exemple, dans (Klaudel, Koutny, Pelz & Pommereau, 2010), les auteurs proposent une technique d'abstraction pour générer des représentations d'espaces d'états réduites pour les systèmes multi-thread. Dans le domaine de la biologie, on trouve le modèle *kappa* (Danos, Feret, Fontana & Krivine, 2008) ainsi que les travaux de thèse de Loïc Paulevé (2011).

Les travaux de Loïc Paulevé (Paulevé, 2011) qui ont été par la suite enrichis par Maxime Folschette (Folschette, 2014), fournissent une approche très spécifique qui repose sur une interprétation abstraite des comportements concurrents des réseaux d'automates (Folschette, Paulevé, Magnin & Roux, 2013). Ils déterminent des représentations abstraites, appelées Graphes de Causalité Locale (GCL), de l'ensemble des comportements nécessaires pour la propriété d'atteignabilité recherchée. Les abstractions calculées ne prennent pas en compte une partie de l'information sur l'ordre ou l'arité des transitions locales. Il en résulte ainsi des approximations supérieures et inférieures des comportements du modèle concret. Une analyse du GCL permet d'identifier les propriétés qui sont soit nécessaires, soit suffisantes à l'atteignalibilité étudiée.

Cette méthode a l'avantage d'avoir une complexité bien inférieure aux méthodes de vérification formelle exacte. En effet, les approches exactes sont de complexité exponentielle selon le nombre d'états dans un seul automate et polynomiale selon le nombre d'automates dans le réseau d'automates asynchrones. Cependant, il existe un risque d'obtenir une réponse non concluante pour le modèle concret, nécessitant alors un raffinement de l'analyse de la dynamique. Une partie de cette analyse statique a par la suite été utilisée dans le but d'approximer efficacement des ensembles de coupes (*cut-sets*), c'est-à-dire des ensembles d'états locaux nécessaires à une certaine accessibilité (Paulevé, Andrieux & Koeppl, 2013). Son utilisation dans ce cadre s'est avérée efficace sur des modèles de plusieurs milliers de composants.

Louis Fippo-Fitime s'est aussi servi par la suite dans (Fippo-Fitime, 2016), des résultats de cette analyse statique pour le développement de nouvelles méthodes d'analyse statique des propriétés quantitatives et pour l'identification des bifurcations (Fippo-Fitime, Roux, Guziolowski & Paulevé, 2017) dans les réseaux d'automates.

Analyses dynamiques du modèle

Les analyses dynamiques se basent plutôt sur des approches exhaustives et permettent habituellement de vérifier la concordance entre un comportement observé du système et une propriété exprimée, par exemple, en logique temporelle (comme CTL (Clarke & Emerson, 2008)), stochastique (comme CSL (Bryans, Bowman & Derrick, 2003)) ou temporisée (comme TCTL (Alur, Courcoubetis & Dill, 1990)).

Parmi les avantages de ces analyses figure le fait qu'elles permettent d'exprimer de nombreux types de comportements ; elles ont potentiellement un grand champ d'application. En plus, les algorithmes de vérification peuvent être génériques : il suffit de savoir générer les transitions possibles à partir d'un état du modèle.

En revanche, l'utilisation de ces méthodes d'analyse est intrinsèquement coûteuse en temps d'exécution ainsi qu'en mémoire surtout quand le nombre d'états est grand (Schnoebelen, 2002). Ainsi, plusieurs travaux ont été menés afin de compresser la mémoire requise pour de telles analyses par l'utilisation des représentations symboliques de l'espace d'états. Ceci est fait par exemple via les diagrammes de décision (notamment, les BDD, *Binary Decision Diagrams* (Bryant, 1986)) qui peuvent être utilisés hiérarchiquement (Couvreur & Thierry-Mieg, 2005; Hamez, Thierry-Mieg & Kordon, 2009).

Dans la pratique, ces techniques permettent d'avoir un gain considérable sur le temps d'exécution des vérifications des modèles ayant un très grand nombre d'états mais leur performance peut dépendre, d'une part, de paramètres liés à leur représentation symbolique (dans le cas des BDD par exemple, l'ordre des composants de l'état impacte fortement leur efficacité), et, d'autre part, de leur complexité théorique.

Analyse des propriétés quantitatives

L'intégration d'une composante temporelle dans les modèles représentant les RRB a ouvert des nouvelles pistes de recherche pour développer des analyses quantitatives dans les RRB. Parmi ces analyses, des travaux proposent des techniques de vérification qui se veulent quantitatives (calculer une probabilité, un délai).

Des travaux ont été initiés pour faire du model checking quantitatif sur des systèmes abstraits sous forme de chaînes de Markov, comme dans (Hansson & Jonsson, 1994; Courcoubetis & Yannakakis, 1988) qui se sont focalisés sur des systèmes à temps discret. Nous citons aussi les travaux de (Bertrand, Bouyer, Brihaye & Markey, 2008) où les auteurs proposent de faire du model checking quantitatif sur les automates temporisés. Ainsi, il est possible de calculer la probabilité d'une propriété régulière ω en corrigeant une abstraction par chaîne de Markov précédemment introduite dans (Baier, Bertrand, Bouyer, Brihaye & Grosser, 2008).

Le formalisme CSL (Continuous Stochastic Logic) a été introduit dans (Zhang, Jansen, Nielson & Hermanns, 2011), afin d'exprimer les propriétés des systèmes abstraits comme des chaînes de Markov à temps continu. Ils ont prouvé que le problème de vérification des propriétés quantitatives est décidable. Il est à noter que CSL est une logique inspirée par la

logique temporelle CTL (Emerson, 1990) et ses extensions pour les systèmes stochastiques à temps discret (Hansson & Jonsson, 1994) et les systèmes non stochastiques à temps continu (Alur et al., 1990).

Finalement, une réduction exacte des modèles stochastiques à base de règles a été proposée dans (Feret, Koeppl & Petrov, 2013) par une abstraction qui permet de réduire l'espace d'états pour les réseaux des interactions protéine-protéine. Cette réduction est basée sur la construction des classes d'équivalence du réseau qui conservent la propriété markovienne. Ces résultats ont été étendus dans (Feret et al., 2012) pour une construction des classes d'équivalence qui conserve la propriété markovienne forte ; c'est-à-dire il existe une forme d'indépendance (une indépendance conditionnelle) entre les états passés et les états futurs.

2.3.2 Les attracteurs

Compte tenu du paradigme utilisé, le comportement à long terme de la dynamique des RRB est d'un intérêt particulier (Wuensche, 1998). En effet, à tout moment, un système peut tomber dans un *domaine piège*, qui fait partie de sa dynamique et il ne peut plus s'en échapper. Lors de l'évolution, le réseau peut éventuellement tomber dans un nouveau et plus petit domaine piège, réduisant ainsi ses comportements futurs possibles (par exemple les états précédents deviennent inaccessibles). Nous appelons alors les domaines pièges minimaux des *attracteurs*. Ces derniers peuvent être singleton (i.e., des points fixes) ou non singletons et illustrant une dynamique cyclique (c'est-à-dire que le réseau oscille ou cycle indéfiniment dans cet ensemble d'états).

Ce phénomène peut évoquer des perturbations biologiques ou d'autres phénomènes complexes. Ainsi, pour toute condition initiale et à long terme, tout réseau finira par atteindre un état final (ou un ensemble d'états finaux) dans lequel les comportements futurs possibles sont plus restreints. Le réseau atteint alors un domaine piège minimal. De tels comportements ont été interprétés comme étant des réponses distinctes et spécifiques de l'organisme, telles que : la différenciation en types cellulaires distincts dans les organismes multicellulaires (Huang, Eichler, Bar-Yam & Ingber, 2005) et la distinction du développement floral normal de la plante Arabidopsis thaliana (Demongeot, Goles, Morvan, Noual & Sené, 2010).

Les conjectures de René Thomas

Les conjectures de René Thomas (Thomas, 1981) sont un exemple d'analyse statique très efficace du graphe des interactions dont le résultat peut se lire directement sur le graphe. En effet, elles tracent un lien entre la présence de circuits dans le graphe d'états et celles d'oscillations ou d'états stables. Nous notons qu'il existe deux types de circuits : (a) des circuits positifs qui contiennent un nombre pair de régulations négatives, (b) et des circuits négatifs qui contiennent un nombre impair de régulations négatives. Ces conjectures sont énumérées ci-dessous :

1. l'existence de plusieurs états stables (ou points fixes) requiert la présence d'un *circuit positif* dans le graphe des interactions ;

2. l'existence d'oscillations soutenues requiert la présence d'un *circuit négatif* dans le graphe des interactions.

La première conjecture a notamment été démontrée dans le cadre de la modélisation booléenne (c'est-à-dire $\forall a \in N$, $l_a = 1$ avec $(N ; E)$ est un graphe des interactions) dans (Remy, Ruet & Thieffry, 2008; Richard, 2006), puis pour la modélisation multivaluée (c'est-à-dire $\exists a \in N$, $l_a > 1$) dans (Richard & Comet, 2007).

La seconde conjecture a également été démontrée dans le cas asynchrone dans le cadre booléen (Remy et al., 2008) et multivalué (Richard, 2010) : la présence d'oscillations soutenues implique la présence d'un circuit négatif dans le graphe des interactions. Un corollaire de cette propriété est que l'absence de circuit négatif implique la présence d'au moins un point fixe dans la dynamique.

L'identification des attracteurs

Le problème du calcul de tous les attracteurs dans un RRB est difficile. Même le problème le plus simple, celui de décider si le système a un point fixe (qui peut être considéré comme le plus petit attracteur) est NP-complet (Zhang, Hayashida, Akutsu, Ching & Ng, 2007). En se basant sur ce résultat, d'autres études ont prouvé que la recherche des attracteurs cycliques (i.e., non singletons) est aussi NP-complet (Klemm & Bornholdt, 2005; Akutsu, Kosub, Melkman & Tamura, 2012).

Bien que certaines méthodes existent avec une complexité inférieure, consistant par exemple à choisir de manière aléatoire un état initial et à suivre une trajectoire suffisamment longue, dans l'espoir de trouver éventuellement un attracteur, elles ne sont pas exhaustives et peuvent manquer des attracteurs (difficiles à atteindre). Nous présentons ainsi, dans la suite quelques travaux développés pour résoudre ce problème d'identification des attracteurs dans les RRB.

Le moyen le plus simple de trouver les attracteurs est d'énumérer tous les états possibles et d'exécuter la simulation de chacun jusqu'à ce qu'un attracteur soit atteint (Somogyi & Greller, 2001). Cette méthode garantit que tous les attracteurs soient détectés mais présente une complexité temporelle exponentielle et, par conséquent, son applicabilité est fortement limitée par la taille du réseau c'est-à-dire son nombre de nœuds).

Dans le cadre des réseaux booléens (Boolean Networks, BN), les algorithmes de détection des attracteurs ont été largement étudiés dans la littérature. Nous citons les travaux de (Irons, 2006), qui propose d'analyser les états partiels afin d'identifier plus efficacement les attracteurs potentiels. Cette méthode améliore l'efficacité du calcul en passant du temps exponentiel à un temps polynomial pour un sous-ensemble de modèles biologiques fortement dépendants de la topologie (les composants prédécesseurs, successeurs et les fonctions de mise à jour) du réseau. Une autre méthode, appelée GenYsis (Garg, Mendoza, Xenarios & DeMicheli, 2007), dont l'algorithme commence à partir d'un état initial (choisi de manière aléatoire) et détecte les attracteurs en calculant ses successeurs et ses prédécesseurs. Cela fonctionne bien pour les petits BN, mais devient inefficace pour les BN de grande taille.

En général, l'efficacité et la capacité du passage à l'échelle des approches d'identification des attracteurs sont encore améliorées grâce à l'intégration de deux techniques. La première est basée sur les diagrammes de décision binaires (Binary Decision Diagrams, BDD), une structure de données compacte pour représenter les fonctions booléennes (la même approche pour la vérification d'atteignabilité). Dans un travail récent (Zhao, Liu, Wang & Qian, 2014), les algorithmes sont basés sur la structure de données BDD réduite (ROBDD), ce qui accélère le temps de calcul de la détection des attracteurs. Ces solutions basées

sur les BDD ne fonctionnent que pour les RRB d'une centaine de nœuds et souffrent du problème d'explosion d'états, car la taille du BDD dépend à la fois des fonctions de régulation et du nombre de nœuds dans les RRB.

L'autre technique consiste à représenter la détection des attracteurs dans des BNs comme un problème de satisfiabilité (SAT) tel que dans (Dubrova & Teslenko, 2011). L'idée principale s'inspire de la vérification de modèle délimitée basée sur SAT : la relation de transition du RRB se déroule en un nombre limité d'étapes afin de construire une formule propositionnelle qui code pour les attracteurs et qui est ensuite résolue par un solveur SAT. Dans chaque étape, une nouvelle variable est nécessaire pour représenter l'état d'un nœud dans le RRB. Il est clair que l'efficacité de ces algorithmes dépend en grande partie du nombre d'étapes de déploiement et du nombre de nœuds dans le RRB (i.e., nombre de composants).

Dans (Mushthofa, Torres, Van de Peer, Marchal & De Cock, 2014), les auteurs séparent la description du réseau (noeuds et leurs interactions : activation ou inhibition) avec les règles de simulation du système (un gène sera activé dans l'état suivant si tous ses activateurs sont actifs ou lorsqu'au moins un de ses activateurs est actif dans l'état actuel). Ils choisissent également le paradigme déclaratif l'Answer Set Programming (ASP) (Baral, 2003) pour avoir une simulation plus flexible, en termes de règles d'évolution d'expression, des modèles de réseau booléen. Ils illustrent que la spécification de grands réseaux avec des comportements plutôt compliqués devient encombrante et est susceptible d'être poussée dans des paradigmes comme SAT, alors que cela est beaucoup moins important dans une démarche déclarative comme la leur.

Un des objectifs de cette thèse est de développer des méthodes exhaustives pour analyser un RRB modélisé avec le formalisme des réseaux d'automates. Nous abordons dans le chapitre 5, deux types de problèmes : trouver tous les points fixes dans un RRB et énumérer tous les attracteurs de taille $n \in \mathbb{N}^*$.

Nous nous concentrons sur deux sémantiques répandues de la dynamique non déterministes (synchrone et asynchrone) et nous utilisons ASP pour résoudre ce problème d'identification des attracteurs. En effet, l'utilisation d'ASP est considérée comme innovante dans le domaine de l'analyse des propriétés dynamiques et notre objectif dans le chapitre 6, est d'évaluer son potentiel de calcul.

2.4 L'apprentissage des RRB à partir des données expérimentales

Le but de la construction des modèles est tout d'abord d'avoir la possibilité d'analyser, de simuler et de comprendre les systèmes (comme il est mentionné dans les sections précédentes de ce chapitre). Nous présentons donc dans cette section un tour d'horizon des travaux qui prennent en compte, dans le processus de modélisation, les données expérimentales décrivant l'évolution d'expression des composants du système biologique. C'est notamment l'objectif du chapitre 4 de cette thèse.

Dans cette démarche, les modèles sont inférés (ou appris) en utilisant une approche d'inférence qui répond au mieux aux questions qui sont posées. Nous présenterons dans la section 2.4.1 quelques techniques d'inférence des modèles à partir des données expérimentales (trouvées par les observations de l'évolution du système à modéliser). Ces données

sont non seulement utilisées pour inférer les modèles, mais aussi pour les réviser et les valider (section 2.4.2).

2.4.1 L'inférence des RRB

Avec la propagation d'outils numériques dans chaque partie de la vie quotidienne et le développement de méthodes NGS (*Next Generation Sequencing methods*) comme les microarrays d'ADN en biologie, une très grande quantité de *données de séries temporelles* est maintenant produite quotidiennement. Cela signifie que les données produites par les expériences sur les systèmes biologiques augmentent considérablement. Les données nouvellement produites, à condition que le bruit associé ne soulève pas de problème par rapport à la précision et la pertinence de l'information correspondante, peuvent nous donner de nouvelles idées sur le comportement d'un système. Cela justifie la motivation de concevoir des méthodes efficaces pour l'inférence des RRB.

En effet, durant ces dernières années, plusieurs recherches ont été menées dans ce domaine d'apprentissage et plus précisément dans les méthodes dites de « l'ingénierie inverse »(*reverse engineering*).

Pour la définition du modèle, divers formalismes de modélisation ont été proposés (section 2.2 en page 25). Ils se différencient par les niveaux de simplifications et les hypothèses émises pour la caractérisation des mécanismes moléculaires entre les composants. Plus généralement, les nœuds du réseaux représentent les composants biologiques du système (les gènes, les protéines, les complexes, etc.). Les interactions entre ces composants dépendent de la méthode d'abstraction des influences. Par conséquent, une caractéristique importante des différentes méthodes d'inférence est le formalisme choisi pour modéliser. Par exemple, dans le cas des BN de (Kauffman, 1969; Thomas, 1973), ils utilisent des variables discrètes $x_i \in \{0, 1\}$ (comme nous l'avons présenté à la section 2.2.2 en page 27) qui définissent l'état du composant (gène, protéine). Ainsi, pour apprendre un tel modèle, pour chaque composant, sa courbe d'expression doit être discrétisée. Plusieurs méthodes sont possibles pour la discrétisation des niveaux d'expression, par exemple, les méthodes de classification (*clustering*) et les méthodes selon les seuils de concentration (présentés dans la suite à la section 4.2.1 du chapitre 4).

Cependant, les approches discrètes qui simplifient le problème d'inférence, par des abstractions doivent déterminer les seuils pertinents de chaque gène pour différencier entre son état actif et son état inactif. Diverses approches ont été conçues pour s'attaquer au problème de la discrétisation. On peut citer par exemple (Zhang, Horimoto & Liu, 2008), dans lequel les auteurs ont défini une méthodologie alternative qui ne considère pas un niveau de concentration (un seuil), mais la façon dont la concentration change (en d'autres termes : la dérivée de la fonction donnant la concentration en fonction du temps) en présence ou absence d'un régulateur. D'autre part, le problème majeur de la modélisation réside dans la qualité des données d'expression fournies. En effet, les données bruitées peuvent être l'origine principale des erreurs dans le processus d'inférence. Ainsi, le prétraitement des données biologiques est crucial pour la pertinence des relations présumées entre les composants.

Dans les BN, les interactions entre les composants sont représentées par des fonctions booléennes. Le défi de l'apprentissage de ce type de modèles, est alors de déterminer ces fonctions booléennes telles qu'elles réussissent à reproduire la dynamique du système illustrée par les observations des données d'expression des gènes. De nombreux algorithmes ont été

proposés dans ce sens comme dans (Liang, Fuhrman & Somogyi, 1998). On cite aussi l'outil CASPO (Guziolowski, Videla, Eduati, Thiele, Cokelaer, Siegel & Saez-Rodriguez, 2013) qui est développé pour générer les modèles logiques (booléens) des signaux de transductions. Cette génération prend en compte les boucles de rétro contrôle.

En revanche, ces méthodes ont de plus le désavantage d'être statiques, c'est-à-dire qu'elles ne permettent pas de modéliser l'évolution du système en fonction du temps. Ainsi, d'autres chercheurs se concentrent plutôt sur l'intégration des aspects temporels dans les algorithmes d'inférence. La pertinence de ces différents algorithmes a récemment été évaluée dans (Koh, Wu, Selvaraj & Kusalik, 2009).

En outre, les auteurs de (Liu, Sung & Mittal, 2004) ont abordé le problème d'inférence de RRB temporisés par le biais de réseaux bayésiens, et dans (Silvescu & Honavar, 2001), les auteurs infèrent un réseau booléen temporel. Puisqu'il s'agit d'un problème complexe, dans (Zhang et al., 2008), les auteurs proposent une technique d'extension de ce qu'ils appellent la fenêtre temporelle (time-window-extension) basée sur la segmentation des séries temporelles en différentes phases successives.

Les avantages de ces méthodes sont leur simplicité et leur faible coût de calcul. De plus, comme elles ne nécessitent pas un gros volume de données (Hecker, Lambeck, Toepfer, Van Someren & Guthke, 2009), elles sont adéquates pour inférer les grands RRB. En revanche, elles ne prennent pas en compte *une régulation à laquelle plusieurs composants participent* comme c'est le cas de la méthode d'inférence présentée dans cette thèse dans le chapitre 4 (tout en gardant l'aspect temporel dans le processus de l'apprentissage).

La révision des modèles existants a fait aussi l'objet de nombreux travaux récents. Quand on parle de révision, on fait référence à l'amélioration du modèle par des nouvelles données fournies : si le modèle n'est pas cohérent avec ces nouvelles données, la révision est faite par la correction (i.e., l'addition ou la suppression) de ses interactions.

Par exemple, dans (Akutsu, Tamura & Horimoto, 2009), les auteurs ont ciblé la révision des BN stationnaires. Cette méthode a été affinée au cours des années. Les travaux récents de (Nakajima & Akutsu, 2013) se concentrent sur la révision des réseaux génétiques variables dans le temps (*Time Varying Genetic Networks*). Ce sont des réseaux dont la topologie ne change pas avec le temps, mais la nature des interactions (activation, inhibition ou absence d'interaction) entre les composants peut changer à certains points temporels (instants finis). L'approche de révision (qui, dans les documents de ces auteurs, est appelée complétion, et se réfère à la fois à l'addition et à la suppression des interactions) a été appliquée avec succès à des études de cas biologiques ; par exemple sur le DREAM4 Challenge (Nakajima & Akutsu, 2014*b*) et sa mise en œuvre a été améliorée par des heuristiques (Nakajima & Akutsu, 2014*a*). Cependant, la méthode est limitée aux réseaux acycliques : c'est-à-dire l'inférence n'est plus possible dans le cas ou il y a des cycles dans le réseau.

Les approches logiques peuvent également être bénéfiques pour la révision des modèles. En effet, elles ont été appliquées avec succès aux réseaux de corrélations (*causal networks*) (Inoue, Doncescu & Nabeshima, 2013) et aux réseaux moléculaires représentés avec le langage SBGN-AF (Yamamoto, Rougny, Nabeshima, Inoue, Moriya, Froidevaux & Iwanuma, 2014).

Notre objectif dans le chapitre 4 de cette thèse, est de fournir une approche logique (son implémentation en ASP est présentée dans le chapitre 6) pour inférer les RRB à partir

des données de séries temporelles. Nous proposons une nouvelle méthodologie pour les modèles représentés par une extension temporelle des réseaux d'automates (un formalisme bien adapté à la modélisation des systèmes biologiques) appelées réseaux d'automates avec le temps (*Timed Automata Networks*, T-AN). Le but principal est alors de réussir à avoir un T-AN résultant aussi cohérent que possible avec les ensembles des données observées.

2.4.2 Validation des RRB appris

Pour les systèmes biologiques, certains modèles prennent plus ou moins bien en compte des propriétés dynamiques des systèmes étudiés. Pour d'autres modèles, par contre, il est nécessaire de les confronter aux données et/ou les enrichir à partir des données pour qu'ils soient plus raffinés voire améliorés. Habituellement, en biologie des systèmes, le traitement consiste à partir d'une base de connaissance de proposer de nouvelles hypothèses de travail. Ces hypothèses vont induire le passage vers les expérimentations. Les résultats des expérimentations sont dans la plupart des cas des informations qui sont traitées afin d'identifier de façon fiable les éléments significatifs et les structures pour le phénomène biologique considéré.

En effet, après l'apprentissage des modèles, il est nécessaire de passer à la validation de ces modèles par d'autres moyens. Ce processus de validation consiste à déterminer si le modèle appris est conforme aux propriétés attendues du système modélisé et aussi aux données expérimentales disponibles sur les observations de l'évolution dynamique du système.

En général, la qualité d'un modèle est déterminée en répondant aux questions suivantes :

- est-ce que le modèle est capable de reproduire la dynamique qu'il avait apprise, et
- est-ce qu'il est capable de prédire correctement les comportements du système qu'il n'a pas appris ?

En tant que modélisateur, il est plus facile de répondre à la première question. En effet, si le modèle appris n'arrive déjà pas à reproduire ce qu'il a appris sur le système qu'il représente il ne pourrait pas prédire ce qu'il n'a pas appris. Par contre, réussir à trouver une réponse à la deuxième question est moins évident. En fait, ceci suppose que le modèle présente idéalement le système réel et donc il peut l'imiter parfaitement. Ce qui n'est pas toujours le cas ; en général les informations disponibles sur le système réel sont incomplètes, bruitées et pas toujours fiables. Une façon de contourner cette difficulté est d'utiliser les données synthétiques avec la conséquence que les performances de la méthode d'inférence du modèle sera fortement liée au modèle utilisé pour calculer ces données. Une autre façon possible est de pré-traiter ces données avant de les utiliser pour apprendre le modèle.

Parmi les approches utilisées, il est à noter celle dans (Klamt, Saez-Rodriguez & Gilles, 2007), où les auteurs proposent des analyses de dépendances entre les composants afin de valider la cohérence d'un graphe des interactions avec des interactions des données expérimentales.

Pour le raffinement d'un modèle qui représente un système biologique, d'autres approches consistent à supprimer les incohérences et/ou à prédire l'information manquante dans les modèles biologiques par la comparaison des attracteurs calculés dans le modèle avec ceux observés. Par exemple, le modèle du développement cellulaire de *Drosophila melanogaster*

a été décrit en utilisant des BN et leurs attracteurs dans (González, Chaouiya & Thieffry, 2008; Albert & Othmer, 2003). Et c'est pareil pour l'étude sur le développement de fleurs dans le *Arabidopsis thaliana* (Mendoza, Thieffry & Alvarez-Buylla, 1999; Demongeot, Morvan & Sené, 2008) et la différenciation T-helper (Abou-Jaoudé, Monteiro, Naldi, Grandclaudon, Soumelis, Chaouiya & Thieffry, 2014).

2.5 RRB via les frappes de processus

Dans cette section, nous définissons les frappes de processus (standards), telles qu'elles ont été introduites par Loïc Paulevé (Paulevé, 2011) et en citant quelques enrichissements parmi ceux qui ont été présentés par Maxime Folschette (Folschette, 2014). Ce formalisme peut être considéré comme simplificateur grâce à sa définition atomique des interactions entre les différents composants du réseau ainsi que sa représentation discrète. Cette simplicité est surtout la principale motivation de son introduction. De ce fait, les frappes de processus sont particulièrement adaptées pour la représentation des RRB. Nous notons que les frappes de processus pourraient aussi bien modéliser des systèmes informatiques concurrents autres que les systèmes biologiques. Ainsi, les résultats introduits dans cette thèse sont aussi applicables à ces systèmes.

2.5.1 Frappes de processus standards

Les frappes de processus (définition 2.9) regroupent un ensemble fini de processus, répartis dans des sortes (ce qui correspond à un composant biologique) : un processus (i.e., niveau discret d'une sorte) appartient à une et une seule sorte. À chaque instant, un et un seul processus de chaque sorte est actif, indiquant l'état courant de la sorte à laquelle il appartient. Le changement de processus actif, d'un instant à un autre, dans une sorte se fait à partir de la frappe du processus actif par au moins un autre processus courant. Un processus est noté a_i où a est la sorte et i l'identifiant du processus au sein de la sorte a.

Les interactions concurrentes entres les processus sont définies par un ensemble d'*actions*. Ces actions permettent le changement d'un processus par un autre de la même sorte.

Nous présentons le formalise des frappes de processus standards (Paulevé, Magnin & Roux, 2011*a*) telles qu'elles sont introduites dans la thèse de Loïc Paulevé (Paulevé, 2011).

Définition 2.9 (Frappes de processus standards)**.** Les *frappes de processus standards* sont définies par un triplet $\mathcal{PH} = (\Sigma\,;\mathcal{L}\,;\mathcal{H})$, où :

- $\Sigma \stackrel{def}{=} \{a, b, \ldots\}$ est l'ensemble fini et dénombrable des *sortes* ;
- $\mathcal{L} \stackrel{def}{=} \prod_{a\in\Sigma} \mathcal{L}_a$ est l'ensemble fini des *états*, où $\mathcal{L}_a = \{a_0, \ldots, a_{l_a}\}$ est l'ensemble fini et dénombrable des *processus* de la sorte $a \in \Sigma$ et $l_a \in \mathbb{N}^*$, chaque processus appartenant à une unique sorte : $\forall (a_i\,;b_j) \in \mathcal{L}_a \times \mathcal{L}_b, a \neq b \Rightarrow a_i \neq b_j$;
- $\mathcal{H} \stackrel{def}{=} \{a_i \to b_j \not\vdash b_k \mid (a\,;b) \in \Sigma\times\Sigma \wedge (a_i\,;b_j\,;b_k) \in \mathcal{L}_a \times \mathcal{L}_b \times \mathcal{L}_b \wedge b_j \neq b_k \wedge a = b \Rightarrow a_i = b_j\}$ est l'ensemble fini des *actions*.

On note **Proc** $\stackrel{def}{=} \bigcup_{a\in\Sigma} \mathcal{L}_a$ l'ensemble de tous les processus. La sorte d'un processus a_i est donnée par $\mathsf{sorte}(a_i) = a$; on définit aussi l'ensemble des sortes d'une action ou d'un

ensemble de processus par :

$$\forall h \in \mathcal{H}, \mathsf{sortes}(h) = \{\mathsf{sorte}(\mathsf{frappeur}(h)), \mathsf{sorte}(\mathsf{cible}(h))\}$$
$$\forall A \subset \mathbf{Proc}, \mathsf{sortes}(A) = \{\mathsf{sorte}(p) \mid p \in A\}$$

Étant donné un état $s \in \mathcal{L}$, le processus de la sorte $a \in \Sigma$ présent dans s est donné par $s[a]$, c'est-à-dire la coordonnée correspondant à a dans l'état s. Si $a_i \in \mathcal{L}_a$, nous définissons la notation : $a_i \in s \overset{def}{\Leftrightarrow} s[a] = a_i$; par extension, si $ps \subset \mathbf{Proc}$, on écrit alors : $ps \subseteq s \overset{def}{\Leftrightarrow} \forall p \in ps, p \in s$. Pour toute action $h = a_i \to b_j \restriction b_k \in \mathcal{H}$, a_i est appelé le *frappeur*, b_j la *cible* et b_k le *bond* de h, et on note : $\mathsf{frappeur}(h) = a_i$, $\mathsf{cible}(h) = b_j$ et $\mathsf{bond}(h) = b_k$.

Exemple. La figure 2.4 illustre une représentation possible des frappes de processus standard. Le modèle $\mathcal{PH} = (\Sigma, \mathcal{L}, \mathcal{H})$ représenté comporte trois sortes : $\Sigma = \{a, b, z\}$. Chaque sorte comporte exactement deux processus :

$$\mathcal{L}_a = \{a_0, a_1\} \quad ; \quad \mathcal{L}_b = \{b_0, b_1\} \quad ; \quad \mathcal{L}_z = \{z_0, z_1\} \quad .$$

On peut notamment en déduire le nombre total d'états du système : $|\mathcal{L}| = |\mathcal{L}_a| \cdot |\mathcal{L}_b| \cdot |\mathcal{L}_z| = 2^3 = 8$. Cette grandeur n'est cependant donnée qu'à titre indicatif, car nous évitons de construire explicitement l'espace des états dont la taille est exponentielle en le nombre de sortes et de processus du modèle. Enfin, le modèle étudié comporte 7 actions :

$$\begin{aligned}\mathcal{H} = \{ \quad & z_1 \to a_1 \restriction a_0 \quad , \quad a_1 \to b_1 \restriction b_0 \quad , \quad b_0 \to z_1 \restriction z_0 \quad , \\ & a_0 \to a_0 \restriction a_1 \quad , \quad z_0 \to b_0 \restriction b_1 \quad , \quad b_1 \to z_0 \restriction z_1 \quad , \\ & \qquad\qquad\qquad\qquad\qquad\quad a_0 \to z_0 \restriction z_1 \qquad \}.\end{aligned}$$

Dans cet exemple, les frappes de processus représentent un modèle simplifié inspiré du mécanisme de segmentation des métazoaires qui permet par exemple de décrire la production de rayures chez les drosophiles. On appelle ce modèle un exemple jouet des frappes de processus.

La définition 2.10 établit la notion de sous-état sur un ensemble de sortes, c'est-à-dire un ensemble de processus qui sont deux à deux de sortes différentes, ce qui permet de ne considérer qu'une partie d'un état complet. L'ensemble de tous les sous-états est noté $\mathcal{L}^\lozenge$ et nous constatons qu'un état est *a fortiori* un sous-état : $\mathcal{L} \subset \mathcal{L}^\lozenge$. De plus nous notons $\mathbf{Proc}^\lozenge$ l'ensemble des sous-états désordonnés, c'est-à-dire dont l'ordre entre les sortes a été oublié.

Le recouvrement d'un état s par un processus a_i est formalisé à la définition 2.11 par un état identique à s, sauf pour le processus de a qui a été remplacé par a_i, ce qui permettra de définir la dynamique des Frappes de Processus à la définition 2.14. La définition de recouvrement est aussi étendue à un sous-état désordonné, autrement dit, à un ensemble de processus contenant au plus un processus par sorte.

Définition 2.10 (Sous-états ($\mathcal{L}^\lozenge$))**.** Si $S \subseteq \Sigma$ est un ensemble de sortes, un sous-état sur S est un élément de : $\mathcal{L}^\lozenge_S \overset{def}{=} \prod_{a \in S} \mathcal{L}_a$. L'ensemble de tous les sous-états est noté : $\mathcal{L}^\lozenge \overset{def}{=} \bigcup_{S \in \wp(\Sigma)} \mathcal{L}^\lozenge_S$.
De plus, si $\sigma \in \mathcal{L}^\lozenge$ et $s \in \mathcal{L}$, on note alors :

$$\sigma \subseteq s \overset{def}{\Leftrightarrow} \forall a_i \in \mathbf{Proc}, a_i \in \sigma \Rightarrow a_i \in s \quad .$$

Enfin, si $S \subset \Sigma$, on note : $\mathbf{Proc}^\lozenge_S = \{\widetilde{ps} \subset \mathbf{Proc} \mid ps \in \mathcal{L}^\lozenge_S\}$ et $\mathbf{Proc}^\lozenge = \{\widetilde{ps} \subset \mathbf{Proc} \mid ps \in \mathcal{L}^\lozenge\}$.

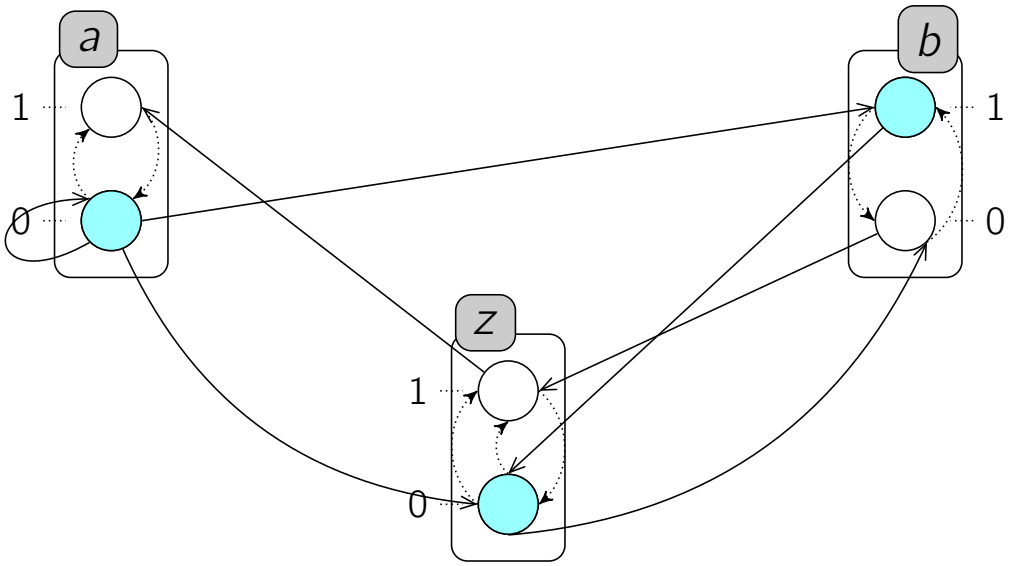


Figure 2.4 : Un exemple de frappes de processus standard. Les sortes sont représentées par des rectangles arrondis contenant des cercles représentant les processus. Ainsi, le processus a_1 est représenté par le cercle marqué « 1 » dans le rectangle étiqueté « a », etc. Chaque action est symbolisée par un couple de flèches, l'une en trait plein et l'autre en pointillés ; par exemple, l'action $a_0 \to z_0 \overset{\curvearrowright}{} z_1$ est représentée par une flèche pleine entre les processus a_0 et z_0 suivie d'une flèche en pointillés entre z_0 et z_1. Enfin, les processus grisés représentent un état courant possible pour ce modèle : $\langle a_0, b_1, z_0 \rangle$ qui peut être par exemple l'état initial.

Définition 2.11 (Recouvrement ($\Cap : \mathcal{L} \times \mathbf{Proc} \to \mathcal{L}$))**.** Étant donné un état $s \in \mathcal{L}$ et un processus $a_i \in \mathbf{Proc}$, $(s \Cap a_i)$ est l'état défini par : $(s \Cap a_i)[a] = a_i \wedge \forall b \neq a, (s \Cap a_i)[b] = s[b]$. On étend de plus cette définition à un ensemble de processus par le recouvrement de l'état par chaque processus, à condition que les processus de l'ensemble soient tous de sortes différentes : $\forall ps \in \mathbf{Proc}^{\Diamond}, s \Cap ps = s \underset{a_i \in ps}{\Cap} a_i$.

2.5.2 Dynamique des frappes de processus

Nous présentons dans la suite les éléments nécessaires pour la dynamique en complément des actions dans les frappes de processus. Le premier élément va être la propriété de jouabilité introduit à la définition 2.12. Cette propriété permet de décrire la présence d'une configuration de processus actifs dans un état donné, ce qui permet de décrire la « jouabilité » d'une action. Aussi la définition 2.13 permet de définir l'*opérateur de jouabilité* des frappes de processus standard. Enfin la dynamique proprement dite est donnée à la définition 2.14. Elle est construite à partir de l'opérateur de jouabilité.

Définition 2.12 (Propriété de jouabilité (F))**.** Une *propriété de jouabilité* est un élément du langage F défini inductivement par :

- $\top$ et $\bot$ appartiennent à F ;
- si $a \in \Sigma$ et $a_i \in \mathcal{L}_a$, alors a_i appartient à F et est appelé un *atome* ;
- si $P \in F$ et $Q \in F$, alors $\neg P$, $P \wedge Q$ et $P \vee Q$ appartiennent à F.

Si $P \in F$ est une propriété de jouabilité et $\sigma \in \mathcal{L}^{\Diamond}$ est un sous-état, on note $[P](\sigma)$ l'*évaluation* de P dans σ :

- si $P = a_i \in \mathcal{L}_a$ est un atome, avec $a \in \Sigma$, alors $[a_i](\sigma)$ est vraie si et seulement si $a_i \in \sigma$;
- si P n'est pas un atome, alors $[P](\sigma)$ est vraie si et seulement si on peut l'évaluer récursivement comme vraie en utilisant la sémantique habituelle des opérateurs $\neg$, $\wedge$ et $\vee$ et des constantes $\top$ et $\bot$.

Une fonction $F : \mathcal{H} \to F$ associant à toute action une propriété de jouabilité est appelée un *opérateur de jouabilité*.

Des propriétés de la logique booléenne sont applicables aux propriétés de jouabilité, à savoir celles concernant la distributivité, l'associativité et la commutativité, ainsi que les lois de De Morgan concernant la négation.

Il en résulte notamment la propriété suivante, permettant d'évaluer la négation d'un atome, et qui dérive naturellement du fait que si un processus n'est pas actif dans un état donné, cela signifie alors qu'un autre processus de la même sorte l'est :

$$\forall a \in \Sigma, \forall a_i \in \mathcal{L}_a, \forall \sigma \in \mathcal{L}^{\Diamond}, [\neg a_i](\sigma) \Leftrightarrow \left[\bigvee_{\substack{a_j \in \mathcal{L}_a \\ a_j \neq a_i}} a_j\right](\sigma)$$

L'opérateur de jouabilité F donné à la définition 2.13 est propre aux frappes de processus standard. En revanche, la dynamique donnée à la définition 2.14 est générale à toutes les frappes de processus, et peut donc à fortiori être utilisée avec l'opérateur F des frappes de processus standard.

Définition 2.13 (Opérateur de jouabilité ($F : \mathcal{H} \to F$))**.** L'opérateur de jouabilité des Frappes de Processus est défini par :

$$\forall h \in \mathcal{H}, F(h) \equiv \mathsf{frappeur}(h) \wedge \mathsf{cible}(h) \ .$$

Définition 2.14 (Dynamique des Frappes de Processus ($\rightarrow_{\mathcal{PH}}$))**.** Une action $h \in \mathcal{H}$ est dite *jouable* dans l'état $s \in \mathcal{L}$ si et seulement si : $[F(h)](s)$. Dans ce cas, $(s \cdot h)$ est l'état résultant du jeu de l'action h dans s, et on le définit par : $(s \cdot h) = s \Cap \mathsf{bond}(h)$. De plus, on note alors : $s \rightarrow_{\mathcal{PH}} (s \cdot h)$.
Si $s \in \mathcal{L}$, un *scénario* δ dans s est une séquence d'actions de $\mathcal{H}$ qui peuvent être jouées successivement depuis s. L'ensemble de tous les scénarios dans s est noté $\mathbf{Sce}(s)$.

Remarque. Les frappes de processus standard possèdent une dynamique totalement asynchrone : entre deux états, une unique action peut être jouée. Ce choix de conception est largement inspiré du modèle de Thomas dont la dynamique est aussi totalement asynchrone. Biologiquement, une telle hypothèse est cohérente avec le fait que la probabilité que deux

composants franchissent simultanément un seuil d'expression est très faible. Cependant, cette sémantique asynchrone permet aussi de simplifier la dynamique des Frappes de Processus standards en assurant que deux états successifs ne varient que d'un seul processus (en fait : exactement d'un processus). Cela a notamment permis le développement de l'analyse statique (Paulevé et al., 2014; Folschette, 2014)

Exemple. La dynamique des frappes de processus standard de la figure 2.4 est représentée à la figure 2.5. On peut notamment y observer le comportement stationnaire normal du modèle qui consiste en une oscillation alternée des processus actifs des sortes a et z :

$$\langle a_0, b_1, z_0\rangle \to_{\mathcal{PH}} \langle a_1, b_1, z_0\rangle \to_{\mathcal{PH}} \langle a_1, b_1, z_1\rangle \to_{\mathcal{PH}} \langle a_0, b_1, z_1\rangle \to_{\mathcal{PH}} \langle a_0, b_0, z_1\rangle \to_{\mathcal{PH}}$$

$$\langle a_0, b_0, z_0\rangle \to_{\mathcal{PH}} \langle a_1, b_0, z_0\rangle \to_{\mathcal{PH}} \dots$$

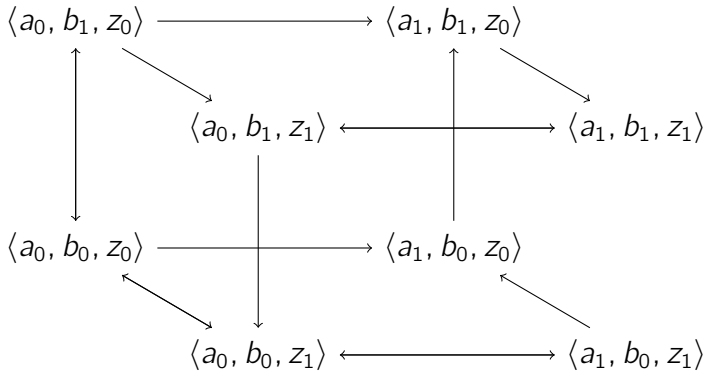


Figure 2.5 : Représentation de la dynamique du modèle de Frappes de Processus standards de la figure 2.4. Chaque état est représenté par un triplet $\langle a_i, b_j, z_k\rangle$ où i, j et k représentent respectivement le niveau d'expression de a, b et z. Une transition entre deux états est représentée par une flèche.

2.5.3 Les enrichissements des Frappes de Processus

Le formalisme des frappes de processus a subi plusieurs enrichissements au cours des dernières années par (Folschette, 2014) et (Fippo-Fitime, 2016). Ainsi, c'est ce qui nous amène, dans cette thèse, à travailler avec les réseaux d'automates qui est une extension des frappes de processus. Nous montrons ainsi dans la suite ces différents enrichissements.

Utilisation des sortes coopératives :

L'une des questions qui se posent en présence d'un formalisme totalement asynchrone comme les frappes de processus standard est la représentation des coopérations entre les différents composants. En effet, le bond d'un processus dans un modèle de frappes de processus standard ne peut se faire que par le jeu d'une action, elle-même déclenchée par la présence d'au plus deux processus : le frappeur et la cible (c'est-à-dire le processus qui va bondir vers un autre processus). Il n'est donc pas possible de conditionner le bond d'un processus par la présence de plusieurs processus de sortes différentes de celle de la

cible. En effet, si on souhaite par exemple représenter l'activation d'un processus c_1 (d'une sorte c) uniquement lorsque deux autres processus a_1 et b_1 (de deux autres sortes a et b) sont actifs, il n'est pas suffisant d'ajouter deux actions $a_1 \to c_0 \Rsh c_1$ et $b_1 \to c_0 \Rsh c_1$, car celles-ci permettent d'activer c_1 à la condition que a_1 soit actif ou que b_1 soit actif : il s'agit bien de deux interactions distinctes et non d'une coopération.

Exemple. Reprenons l'exemple jouet, le modèle de Frappes de Processus de la figure 2.4 en page 44. Dans ce modèle, la production ou l'activation du composant z devrait uniquement être possible à la condition suivante : « b est actif et a n'est pas actif ». Or dans l'état courant du modèle, en cas de désactivation de a, la désactivation du gène b n'empêche pas la production de z car depuis tout état contenant b_0 et a_0, il est toujours possible d'activer z à l'aide des actions $a_0 \to z_0 \Rsh z_1$.

Afin de représenter la coopération entre composants, et donc le raffinement de la dynamique des modèles, Paulevé et al. (2011*a*) ont proposé d'ajouter des sortes particulières appelées *sortes coopératives*, qui servent exclusivement à la modélisation. Une sorte coopérative permet de représenter l'état conjoint de plusieurs sortes dans le modèle. Pour cela, à chaque processus de la sorte coopérative correspond un sous-état des sortes qu'elle *représente*. Ainsi, il est possible de représenter les différents états combinés d'un ensemble de sortes, afin de ne jouer une action que dans une configuration particulière. Ces sortes ont l'avantage d'être des sortes standards, et donc de ne pas nécessiter d'enrichissement particulier de la sémantique. De plus, leur utilisation n'impacte pas les méthodes d'analyse de la dynamique développées : en effet, ces méthodes sont principalement impactées par le nombre de processus dans chaque sorte, et non le nombre total de sortes. Ainsi, à condition de limiter le nombre de processus dans les sortes coopératives, comme expliqué à la fin de cette section, il est possible de les utiliser en maintenant de bonnes performances d'analyse.

Exemple. Les frappes de processus standards de la figure 2.6 ci-dessous reprennent les trois sortes a, b et z du modèle de la figure 2.4 et comprennent en plus une sorte coopérative ab permettant de détecter la présence simultanée de a_0 et b_1. Les processus de ab décrivent les combinaisons possibles des états de a et de b : ab_{00} correspond à a_0 et b_0, ab_{01} correspond à a_0 et b_1, etc. Les actions en amont de cette sorte coopérative permettent de la mettre à jour, c'est-à-dire de changer son processus actif en fonction des évolutions du processus actif de a et b. Par exemple, si a_1 est actif, les deux actions $a_1 \to ab_{00} \Rsh ab_{10}$ et $a_1 \to ab_{01} \Rsh ab_{11}$ effectuent cette mise à jour en faisant bondir le processus actif de ab depuis un processus représentant la présence de a_0 (ab_{00} ou ab_{01}) vers le processus correspondant représentant la présence de a_1 (respectivement ab_{10} ou ab_{11}). Son action en aval, $ab_{01} \to z_0 \Rsh z_1$, joue alors le rôle d'une coopération entre a_0 et b_1 pour frapper z_0 et le faire bondir en z_1.

Exemple. Afin de pallier partiellement l'absence de coopération entre les sortes a et b dans le modèle de la figure 2.4 en page 44, il est possible d'intégrer la sorte coopérative décrite à la figure 2.6. La figure 2.7 propose un modèle corrigé de cette manière, avec une sorte coopérative ab permettant de détecter la présence de a_0 et b_1 . Les deux actions $a_0 \to z_0 \Rsh z_1$ et $b_1 \to z_0 \Rsh z_1$ sont alors remplacées par une action $ab_{01} \to z_0 \Rsh z_1$ afin d'avoir une véritable coopération entre ces deux processus pour activer z.

Nous notons pour terminer qu'il existe deux manières de diminuer le nombre de processus dans une sorte coopérative.

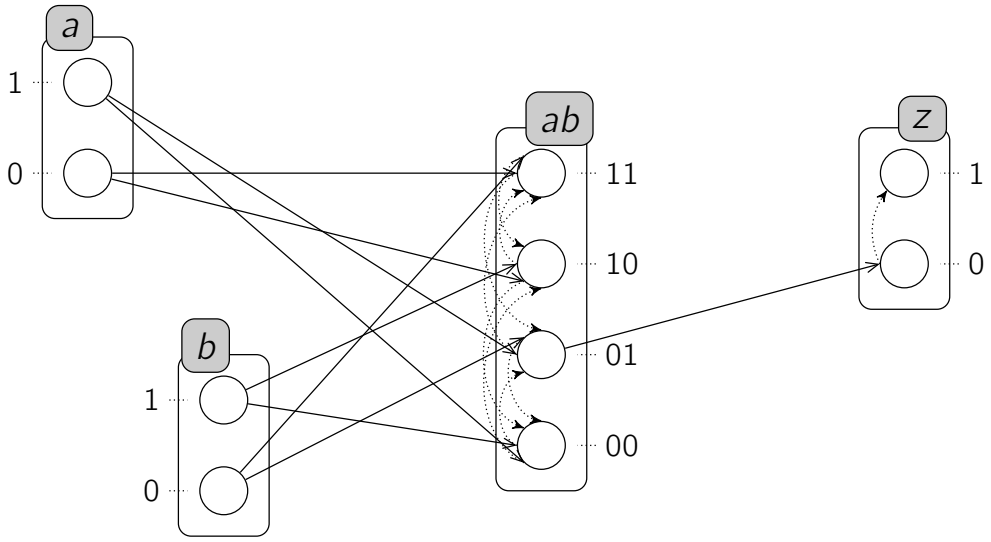


Figure 2.6 : Un exemple de frappes de processus standard avec une sorte coopérative ab. L'action $ab_{01} \rightarrow z_0 \Rsh z_1$ modélise une coopération entre a_0 et b_1 pour faire bondir le processus actif de z.

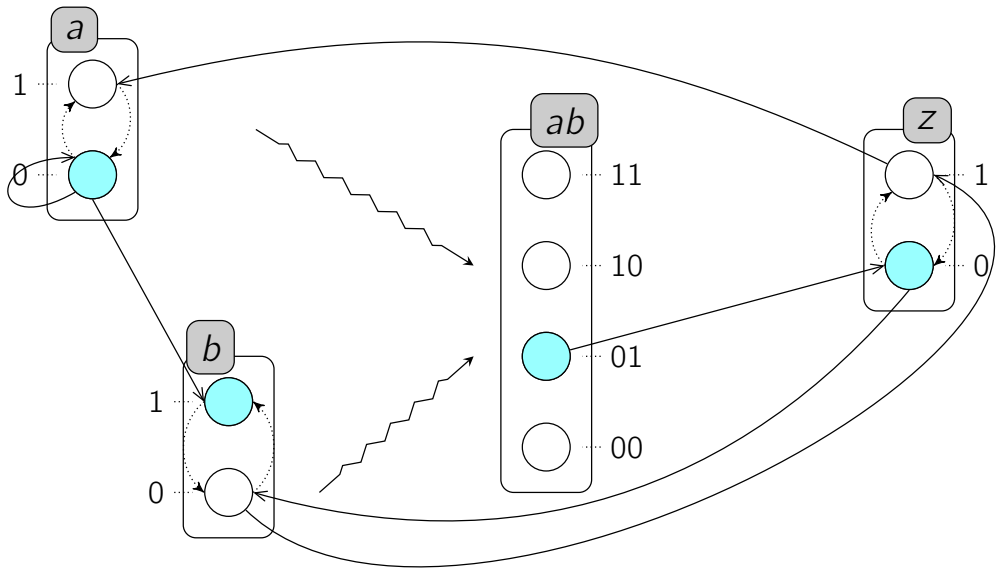


Figure 2.7 : Amélioration du modèle de Frappes de Processus de la figure 2.4 à l'aide de la sorte coopérative ab. Les processus de cette sorte représentent les différents sous-états formés par les deux sortes a et b. Ainsi, ab_{00} représente le fait que a_0 et b_0 sont actifs, etc. Les actions permettant la mise à jour de cette sorte coopérative n'ont pas été représentées explicitement ici mais sont symbolisées par les deux flèches en zigzag provenant de a et b et leur construction est immédiate.

On peut tout d'abord se limiter à deux états par sorte représentée : un état « bon » et un état « mauvais », même si les sortes représentées possèdent plus de deux processus. Cela permet de limiter le nombre de processus de la sorte coopérative à $2^{|A|}$, où A est l'ensemble des sortes à représenter, bien que le nombre de processus dépende toujours exponentiellement de la taille de A.

Il est aussi possible de factoriser les sortes coopératives qui représentent trois sortes ou plus. Pour cela, il suffit par exemple de créer une sorte coopérative intermédiaire entre deux des sortes représentées, puis une deuxième sorte coopérative entre la troisième sorte représentée et cette nouvelle sorte. Ainsi, trois sortes a, b et c peuvent être représentées soit par une unique sorte coopérative abc, soit par deux sortes coopératives ab et abc, la seconde représentant en fait les sortes ab et c (et non a, b et c directement). Le nombre de processus requis devient alors polynomial dans la taille de l'ensemble des sortes à représenter. En conjonction avec la méthode précédente, le nombre de processus requis devient effectivement $4 \cdot (|A| - 1)$.

Les sortes coopératives possèdent cependant un inconvénient, qui est lié au fait que les actions sont totalement indépendantes car le formalisme des frappes de processus standard est totalement asynchrone et non-déterministe. En effet, les sortes coopératives ne sont pas nécessairement mises à jour immédiatement après un bond du processus actif d'une des sortes qu'elles représentent. Il peut ainsi exister un « décalage temporel » entre le changement de processus actif d'une sorte et la mise à jour des sortes coopératives. Ce décalage temporel permet alors de jouer des actions modélisant des coopérations dans des états où la coopération n'est plus possible, car même si l'un des processus modélisant la coopération a bondi, la sorte coopérative peut ne pas en avoir fait de même. Mais ce décalage peut aussi aboutir à des comportements indésirables. En effet, le processus actif d'une sorte coopérative ne correspond de fait pas à l'état courant des sortes représentées, mais uniquement à une combinaison d'états passés. Il est alors possible d'activer un processus de la sorte coopérative correspondant à un sous-état artificiel, c'est -à-dire non accessible aux sortes représentées.

Exemple. Malgré l'ajout d'une sorte coopérative ab dans le modèle de la figure 2.4, il faut noter que le comportement désiré n'est pas exactement représenté. En effet, l'ajout de cette sorte coopérative devait permettre d'éviter toute activation de z lorsque b devenait inactif (et a est inactif aussi), en permettant par exemple de jouer ce type de scénario depuis l'état initial $\langle a_0, b_1, z_0, ab_{01} \rangle$:

$$a_0 \to b_1 \Rsh b_0 :: b_0 \to ab_{01} \Rsh ab_{00}$$

après lequel il n'est plus possible d'atteindre un état où z_1 est actif.

Cependant, il se trouve qu'il existe encore un cas particulier où a peut être activé malgré la présence de b_0. Ce cas particulier relève du comportement mis en valeur ci-dessus, où une action a pour frappeur un processus de sorte coopérative qui ne devrait pas être actif si celle-ci avait été mise à jour. Il s'observe par exemple en jouant le scénario suivant depuis l'état initial $\langle a_0, b_0, z_1, ab_{01} \rangle$:

$$a_0 \to b_1 \Rsh b_0 :: ab_{01} \to z_0 \Rsh z_1 \ ,$$

ce qui est possible parce que la sorte ab n'a pas été mise à jour avant le jeu de l'action $ab_{01} \to z_0 \Rsh z_1$.

Frappes de processus avec actions plurielles :

Afin de mieux prendre en compte un système au niveau de ses réactions biochimiques, c'est-à-dire des réactions entre les différents composants présents, (Folschette, 2014) a proposé un autre enrichissement des frappes de processus standard qui sont les *frappes de processus avec actions plurielles*. Ces réactions peuvent avoir différentes formes (transformation, complexation, dissociation...), et il est fréquent qu'elles fassent intervenir plusieurs réactifs et plusieurs produits. Cette extension va dans le même sens que la sémantique booléenne de Biocham (Fages, Soliman & Chabrier-Rivier, 2004) qui propose par exemple de modéliser un tel système de réactions biochimiques à l'aide d'un ensemble de règles de réaction de la forme : $X \xrightarrow{Y} Z$, ou encore : $X + Y \to Y + Z$, où X est un ensemble de réactifs, Y un ensemble de catalyseurs et Z un ensemble de produits.

Les *frappes de processus avec actions plurielles* permettent de représenter de telles réactions mettant en jeu un nombre arbitraire de réactifs, de produits et de catalyseurs. Ainsi, une réaction de la forme : $X \xrightarrow{Y} Z$ peut être représentée à l'aide de l'action $A \rightarrowtail B$ où A et B sont deux ensembles des processus, A regroupant tous les processus représentant les composants nécessaires à initier la réaction, et B tous les processus qui ont évolué pendant la réaction. Une telle action peut donc être jouée dans un état contenant tous les processus de A et fait évoluer celui-ci vers un état contenant tous les processus de B, les autres processus restant inchangés. Cela implique toutefois que pour tout processus de B, il existe un autre processus de la même sorte dans A. Les frappes de processus avec actions plurielles permettent donc de représenter un nombre arbitraire de bonds simultanés — autrement dit, de changements simultanés de processus actifs — déclenchés par un nombre arbitraire de prérequis — sous la forme de processus actifs dans l'état courant.

Un parallèle peut être tracé d'une part entre A et l'ensemble des réactifs et catalyseurs, et d'autre part entre B et l'ensemble des produits. Cependant, la modélisation par frappes de processus avec actions plurielles nécessite aussi de donner explicitement les composants qui sont absents. Par exemple, une réaction de complexation du type : $x + y \to c$ est représentée en frappes de processus avec actions plurielles à l'aide de trois sortes x, y et c contenant chacune deux processus et représentant respectivement les deux réactifs et le complexe produit, et par l'action $\{x_1, y_1, c_0\} \rightarrowtail \{x_0, y_0, c_1\}$.

Autrement dit, il est nécessaire de décomposer chaque élément en fonction de sa présence (x_1 et y_1) ou de son absence (c_0) au début comme à la fin de la réaction, et pas uniquement d'indiquer les composants présents en tant que réactifs ou produits.

On note ainsi qu'une réaction de la forme $\{a_0, b_0, c_0\} \rightarrowtail \{a_1, b_1\}$ ne peut pas être jouée si a ou b est déjà au niveau 1, comme c'est le cas par exemple dans l'état $\langle a_1, b_0, c_0 \rangle$. Un tel comportement a du sens lorsque les différents processus d'une sorte (a_0 et a_1, par exemple) représentent différents états d'une même molécule : la réaction ne peut alors pas être jouée pour des raisons de stœchiométrie. Cependant, si ces différents processus représentent plutôt des niveaux de concentration (a_1 représentant par exemple un niveau de concentration de la molécule a plus élevé que a_0), cette restriction n'a plus de sens car une plus forte concentration d'une des entités ne devrait pas empêcher la réaction d'avoir lieu et de produire la seconde entité. Cela peut néanmoins être corrigé en ajoutant les actions $\{a_1, b_0, c_0\} \rightarrowtail \{a_1, b_1\}$ et $\{a_0, b_1, c_0\} \rightarrowtail \{a_1, b_1\}$, ou encore en séparant la production de a_1 et de b_1 en deux actions (ou ensembles d'actions) distinctes.

Exemple. Dans le but d'améliorer la modélisation en frappes de processus et de n'avoir que des comportements plus réalistes, nous proposons dans cet exemple de la figure 2.8 un remplacement de la sorte coopérative de l'exemple de la figure 2.6 par une action plurielle. La sorte coopérative ab et toutes ses actions (entrantes et sortantes) notamment l'action $ab_{01} \to z_0 \Rsh z_1$ sont remplacées par l'action plurielle : $\{a_0, b_1\} \to z_0 \Rsh z_1$ qui permet de garder la coopération entre les deux processus a et b pour activer z et aussi de simplifier le visuel du modèle et de ne pas avoir besoin d'un recours vers une nouvelle sorte dans le modèle.

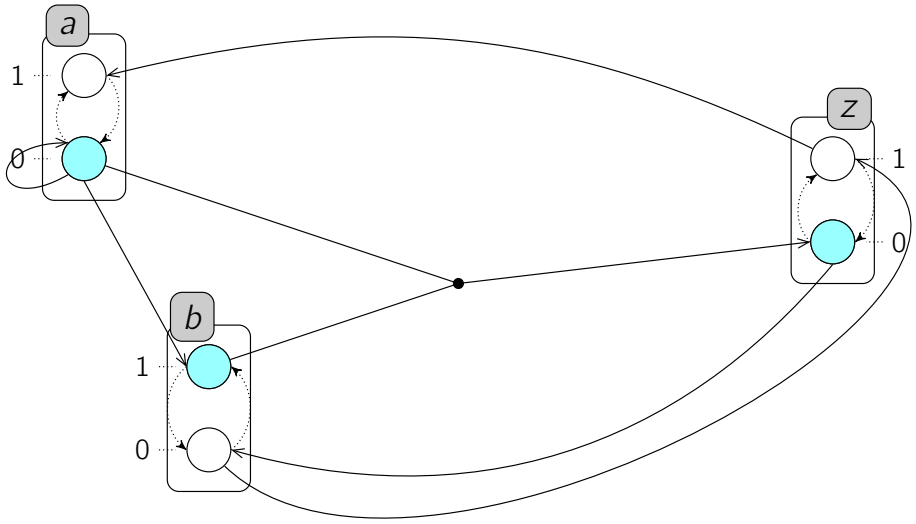


Figure 2.8 : Amélioration du modèle de frappes de processus de la figure 2.4 en page 44 à l'aide de l'action plurielle $\{a_0, b_1\} \to z_0 \Rsh z_1$. Ainsi cette action plurielle est jouable dans un état que lorsque a_0 et b_1 sont actifs.

2.5.4 Discussion

La forme des frappes de processus peut être aisément représentée à l'aide d'un réseau d'automates synchronisés, car chaque sorte possède un rôle similaire à celui d'un automate et chaque action pouvant être remplacée par un ensemble de transitions étiquetées avec le même libellé. Plus généralement, les frappes de processus avec actions plurielles peuvent être considérées comme une restriction des réseaux d'automates stochastiques introduits par (Plateau & Atif, 1991) pour la modélisation des systèmes parallèles.

Dans cette thèse, nous nous concentrons sur l'utilisation du formalisme des réseaux d'automates (AN, *Automata Networks*) (Folschette et al., 2015; Paulevé, 2016*a*), et qui englobe notamment le cadre des frappes de processus avec actions plurielles. En effet, comme il a été indiqué dans ce chapitre, les modèles qualitatifs ont reçu une attention

considérable, en raison de leur capacité à saisir les comportements biologiques complexes efficacement grâce à des abstractions.

Néanmoins, l'originalité de notre contribution est de considérer les modèles AN : ce formalisme permet aux entités d'avoir des niveaux d'expression booléens (1 pour actif et 0 pour inactif) ou multivalués. Il peut représenter des interactions complexes par des transitions locales dans les automates en relation avec les états des autres automates. Ces transitions locales conditionnelles remplacent alors les transitions plurielles dans les frappes de processus.

Cette approche de modélisation avec le formalisme AN (présentée avec plus de détails dans le chapitre 3 suivant) permet de représenter une large gamme de modèles dynamiques. En outre, la forme particulière de ses transitions locales peut être bien gérée en le langage de la programmation logique, Answer Set Programming (ceci est montré dans le chapitre 6). Enfin, ce cadre permet de représenter aussi des modèles synchrones non déterministes, contrairement aux travaux précédents sur l'analyse de la dynamique (dont quelques uns sont cités dans ce chapitre) qui sont axés sur des modèles asynchrones ou synchrones déterministes. Nous détaillons ceci dans le chapitre 5.

Chapitre 3

Les Réseaux d'automates avec le temps

Nous introduisons dans ce chapitre, une extension du formalisme des réseaux d'automates (AN, *Automata Networks*) appelée les *réseaux d'automates avec le temps* (T-AN, *Timed Automata Networks*). Cette extension consiste à enrichir les AN par l'intégration d'une information temporelle appelée le *délai*. Cette nouvelle composante du temps qui est spécifique à chaque transition locale dans un T-AN, fournit une précision sur la durée nécessaire pour que la transition s'active afin de changer l'état d'un composant. De plus, cet enrichissement permet de restreindre la dynamique générale des AN afin de saisir un comportement plus réaliste. Ainsi, nous développons dans ce chapitre une nouvelle sémantique de la dynamique des T-AN qui est raffinée par rapport à celle des AN.

Ce travail a d'abord été publié dans (Ben Abdallah, Ribeiro, Magnin, Roux & Inoue, 2016) puis une version étendue a été publiée dans (Ben Abdallah, Ribeiro, Magnin, Roux & Inoue, 2017).

3.1 Introduction

L'objectif de ce chapitre est de montrer comment nous introduisons le temps dans les réseaux d'automates (AN) et ainsi de passer des AN vers des réseaux d'automates avec le temps (T-AN).

Le formalisme des AN a été introduit dans une forme restreinte appelée les "Frappes de Processus" dans (Paulevé, 2011) et que nous avons précédemment présentées dans la section 2.5 en page 42 du chapitre 2. Il s'est avéré que le formalisme AN est adéquat pour modéliser les réseaux de régulation biologiques (Paulevé et al., 2014). Un AN modélise les composants du système étudié et leurs interactions. En effet, à partir d'un AN, nous

construisons la séquence des événements discrets qui illustre la dynamique du système modélisé. Cette séquence est une succession des états du réseau et montre seulement l'ordre *chronologique* entre les états sans aucune information sur la durée qui les sépare. Il n'y a pas d'information sur la mesure *chronométrique* du temps passé dans un état avant de changer vers un nouvel état successeur. Ceci est dû au fait que les interactions sont considérées comme instantanées dans un AN. La figure 3.1 ci-dessous illustre la durée de temps à laquelle nous nous intéressons et qui est nécessaire au déclenchement d'une transition.

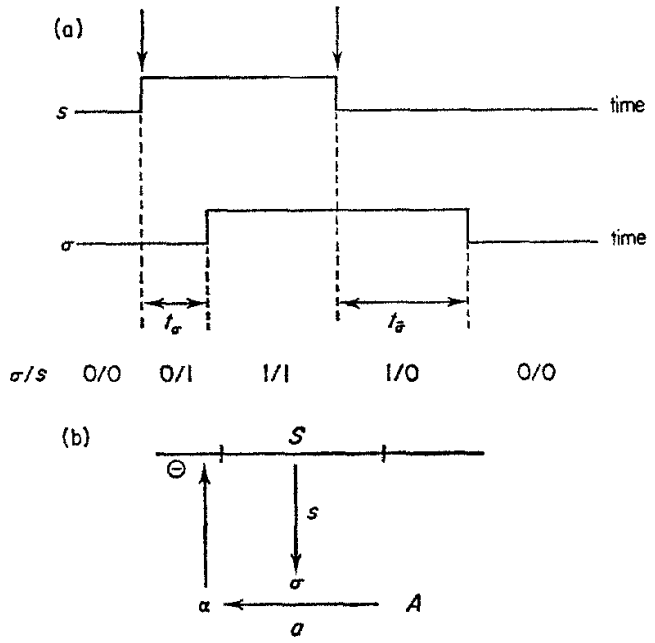


Figure 3.1 : Tirée de (Thomas, 1978) : explication des *délais* dans une interaction entre deux composants S et σ. (a) Si le gène S est activé à l'instant indiqué par la flèche gauche (S change de 0 à 1), le produit σ apparaîtra et atteindra une concentration efficace, mais seulement après un délai d'"établissement" t_σ, (σ change de 0 à 1). Lorsque S est désactivé (flèche à la droite), S tombe de 1 à 0 et σ le suivra après un délai de "dégradation" $t_{\bar{\sigma}}$. Les valeurs successives de σ et S sont = : 0/0, 0/1, 1/1, 1/0 et 0/0. (b) Le gène S s'exprime par une enzyme σ qui catalyse la transformation du métabolite A en α. Le produit α empêche à son tour l'expression du gène S. Ceci est une boucle de rétroaction négative.

Par conséquent, notre but est de raffiner la dynamique des AN par l'introduction d'un aspect temporel comme celui indiqué dans cette figure 3.1. Nous intégrons alors une composante temporelle dans les interactions qui existent entre les composants du système. Cette variable de temps δ (noté par t_σ et $t_{\bar{\sigma}}$ dans la figure 3.1 ci-dessus) est spécifique pour chaque interaction et représente la période nécessaire pour qu'une interaction réalise son effet. Autrement dit, δ est la durée dont la transition a besoin pour que le changement qu'elle cause soit effectif.

Dans les AN, nous représentons une interaction par une transition locale dans un automate (i.e., un composant du système) et cette transition est responsable du changement de l'automate d'un état discret vers un autre. L'activation d'une transition locale (c'est-à-dire elle débute son influence sur le composant qu'elle cible) est conditionnée par les états

d'autres automates du réseau et plus précisément, seulement ceux qui ont une influence sur l'automate ciblé. Nous présentons davantage ce formalisme des AN dans la section 3.2.

Nous introduisons aussi dans ce chapitre l'impact de cet enrichissement sur la dynamique du modèle. En effet, l'ajout de telles informations temporelles peut permettre de privilégier un chemin sur un autre à un moment de l'évolution du modèle. Leur intégration permet donc d'affiner le modèle en réduisant les comportements possibles afin d'obtenir un modèle plus proche du système modélisé. Nous montrons dans la section 3.3 de ce chapitre, que la sémantique de la dynamique des T-AN est beaucoup plus affinée que celle des AN. En fait, la sémantique de la dynamique des AN porte principalement sur l'*asynchrone* ou sur le *synchrone*. La différence entre ces deux dernières réside dans le fait qu'en asynchrone, une seule transition locale peut s'activer entre deux états successifs du réseau, mais en sémantique synchrone, toutes les transitions sont activées en parallèle. Par contre, ce que nous montrons dans ce chapitre, c'est que la dynamique d'un réseau est plutôt une sémantique combinée entre l'asynchrone et le synchrone et que le choix est fait selon l'état du réseau au cours de son évolution dynamique et selon le modèle T-AN lui même. En effet, c'est grâce à l'introduction du temps qu'une telle sémantique s'impose et c'est ce que nous montrons dans la section 3.3 de ce chapitre.

Finalement, nous comparons dans la section 3.4, notre formalisme T-AN par rapport à d'autres formalismes existants qui intègrent aussi une variable du temps et qui présentent des sémantiques différentes. Nous concluons ce chapitre par une discussion dans la section 3.5.

3.2 Les Réseaux d'automates

Nous définissons dans cette section les AN, un formalisme plus enrichi que les Frappes de Processus (présentées dans la section 2.5 en page 42 du chapitre 2). Nous pouvons assimiler les AN à des Frappes de Processus avec des actions plurielles (introduites dans la sous-section 2.5.3 en page 46) et qui ont été introduites dans la thèse de Maxime Folschette en 2014 (Folschette, 2014).

Un AN est considéré simple grâce à sa définition atomique des interactions entre les composants du système qu'il modélise, ainsi que sa représentation discrète. AN est donc considéré comme particulièrement adapté pour la représentation d'un Réseau de Régulation Biologique (RRB).

Différentes sémantiques de la dynamique peuvent être utilisées avec les formalismes des modèles discrets de type AN. Nous introduisons dans cette section les deux sémantiques les plus répandues dans la littérature : l'asynchrone et le synchrone.

3.2.1 Définitions des réseaux d'automates

La définition 3.1 introduit les AN comme un modèle avec des automates ayant un nombre fini d'états discrets appelés *états locaux*. Un *état local* d'un automate a est noté par a_i où a est l'identifiant de l'automate et i le niveau d'expression de a. A chaque instant, chaque automate a exactement un seul état local *actif*. L'ensemble des états locaux actifs de tous les automates est appelé l'*état global* du réseau. Le changement des automates d'un état local actif vers un autre est assuré par les *transitions locales*. Leurs activités sont

conditionnées par les états locaux actifs des autres automates du réseau (voir figure 3.2 ci-dessous).

Définition 3.1 (Réseaux d'automates (AN))**.** Un *Réseau d'Automates* (Automata Network, AN) est un triplet $(\Sigma, \mathcal{S}, \mathcal{T})$ tel que :

- $\Sigma = \{a, b, \ldots\}$ est l'ensemble fini des identifiants des automates ;
- Pour tout $a \in \Sigma$, $\mathcal{S}_a = \{a_i, \ldots, a_j\}$ est l'ensemble fini des *états locaux* de l'automate a ; $\mathcal{S} = \prod_{a\in\Sigma} \mathcal{S}_a$ est l'ensemble fini des *états globaux* ;
  On note par $\mathbf{LS} = \cup_{a\in\Sigma}\mathcal{S}_a$ l'ensemble de tous les états locaux.
- Pour tout $a \in \Sigma$, $\mathcal{T}_a = \{a_i \xrightarrow{\ell} a_j \in \mathcal{S}_a \times \wp(\mathbf{LS} \setminus \mathcal{S}_a) \times \mathcal{S}_a \mid a_i \neq a_j\}$ est l'ensemble des *transitions locales* de l'automate a ; $\mathcal{T} = \bigcup_{a\in\Sigma} \mathcal{T}_a$ est l'ensemble de toutes les transitions locales dans le modèle.

Les interactions entre les automates sont définies par un ensemble de transitions locales. Chaque transition locale a la forme suivante : $\tau = a_i \xrightarrow{\ell} a_j$, où a_i et a_j sont des états locaux de l'automate a appelés respectivement *origine* et *destination* de τ. ℓ est la *condition* de τ et c'est l'ensemble des états locaux des autres automates différents de a (avec au plus un état local par automate). ℓ pourrait être égal à l'ensemble vide s'il s'agit d'une transition spontanée autonome (ou auto-transition).

Pour la transition locale τ, on note alors $\mathsf{ori}(\tau) = a_i$, $\mathsf{dest}(\tau) = a_j$ et $\mathsf{cond}(\tau) = \ell$.

Exemple. La figure 3.2 ci-dessous représente un réseau d'automates $(\Sigma, \mathcal{S}, \mathcal{T})$ avec 4 automates libellés par : a, b, c, et d. Comme le montre cette figure, les automates a et c ont chacun 2 états locaux 0 et 1, mais b et d ont 3 états locaux 0, 1 et 2. Et il y a au total 12 transitions locales. Le triplet $(\Sigma, \mathcal{S}, \mathcal{T})$ du modèle est détaillé textuellement ci-dessous :

- $\Sigma = \{a, b, c, d\}$,
- $\mathcal{S}_a = \{a_0, a_1\}$, $\mathcal{S}_b = \{b_0, b_1, b_2\}$, $\mathcal{S}_c = \{c_0, c_1\}$, $\mathcal{S}_d = \{d_0, d_1, d_2\}$,
- $\mathcal{T} = \{\, a_0 \xrightarrow{\{c_1\}} a_1,\ a_1 \xrightarrow{\{b_2\}} a_0, \quad b_0 \xrightarrow{\{d_0\}} b_1,\ b_0 \xrightarrow{\{a_1,c_1\}} b_2,\ b_1 \xrightarrow{\{d_1\}} b_2,\ b_2 \xrightarrow{\{c_0\}} b_0,$
  $c_0 \xrightarrow{\{a_1,b_0\}} c_1,\ c_1 \xrightarrow{\{d_2\}} c_0, \quad d_0 \xrightarrow{\{b_2\}} d_1,\ d_0 \xrightarrow{\{a_0,b_1\}} d_2,\ d_1 \xrightarrow{\{a_1\}} d_0,\ d_2 \xrightarrow{\{c_0\}} d_0\ \}.$

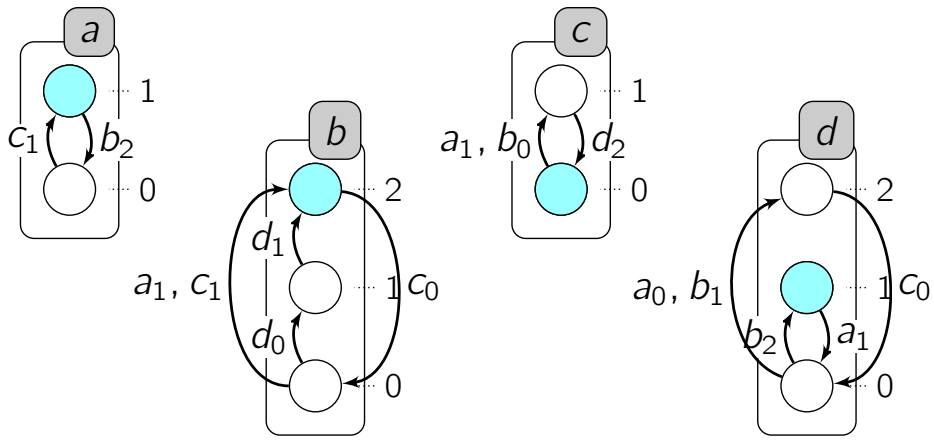


Figure 3.2 : Un exemple d'un modèle de réseau d'automates ayant 4 automates a, b, c et d. Chaque boite représente un automate (modélisant par exemple un composant biologique), les cercles représentent leurs états locaux (correspondant aux niveaux d'expression discrets) et les transitions locales sont représentées par des flèches. Les états locaux colorés en bleu représentent l'état global du réseau $\langle a_1, b_2, c_0, d_1\rangle \sim 1201$ à un instant donné.

Les transitions locales d'un AN définissent les interactions entre les automates. Concrètement, dans un système biologique, ces transitions locales représentent les différentes interactions possibles entre les composants biologiques : *activation* ou *inhibition*.

Par exemple dans la figure 3.2 ci-dessus, la transition locale $a_0 \xrightarrow{\{c_1\}} a_1$ introduit le fait que l'automate (représentant le composant biologique) a est activé (passe du niveau 0 au niveau 1) par l'automate c lorsque ce dernier est activé (i.e., il est au niveau 1 : c_1). D'autre part, a a aussi une rétroaction positive sur c lorsque b est inactif, (i.e., b est au niveau 0). Cette activation est représentée par la transition locale $c_0 \xrightarrow{\{a_1, b_0\}} c_1$.

En ce qui concerne la dynamique générale du réseau, elle est trouvée à partir de l'ensemble des transitions locales qui sont activées successivement. Et par rapport à une sémantique choisie de la dynamique, transitions locales sont activées parallèlement ou indépendamment. En effet, à chaque instant, le réseau a un seul *état global* ; c'est un n-uplet des états locaux actifs de tous ses $n \in \mathbb{N}$ automates. Par exemple, dans la figure 3.2 ci-dessus, le réseau a 4 automates et son état global est $\langle a_1, b_2, c_0, d_1 \rangle$. Calculer tous les états globaux d'un AN ainsi que toutes les *transitions globales* qui permettent d'évoluer d'un état global vers un autre, donne la possibilité de trouver le graphe d'états du réseau. Mais nous allons montrer que des solutions existent pour analyser les modèles sans toujours construire ce graphe complet qui peut être très gros.

Nous présentons dans la figure 3.3 en page 61 une partie du graphe d'états du AN de l'exemple de la figure 3.2 trouvé selon la sémantique asynchrone et dans la figure 3.4 en page 62 le graphe d'états du même réseau calculé selon la sémantique synchrone. Nous donnons plus de détails dans la section suivante sur ces deux sémantiques. En plus, nous expliquons comment une simulation d'un AN est faite afin de construire son graphe d'états.

3.2.2 Sémantique de la dynamique des réseaux d'automates

On trouve dans la littérature plusieurs sémantiques de la dynamique qui ont été proposées, à savoir le synchrone (Kauffman, 1969), l'asynchrone (Thomas, 1991) et même des sémantiques qui combinent les deux par exemple dans (Bridoux, Guillon, Perrot, Sené & Theyssier, 2017). Cette diversité est due au fait que différentes sémantiques peuvent faire évoluer le système vers des attracteurs (états terminaux du système) différents. Les attracteurs que nous étudions dans la section 5.4 en page 146 du chapitre 5, présentent une propriété importante du système étudié et leur identification fait partie de la validation ou de la réfutation d'un modèle. De plus, l'étude de la dynamique d'un système se fait à travers l'étude des propriétés dynamiques du modèle qu'il représente. Comme par exemple, la vérification de l'atteignabilité d'un état à partir d'un état global initial. Et réussir à pouvoir retrouver la dynamique réelle d'un système par son modèle représentatif repose d'une part sur le modèle lui même et d'autre part sur la sémantique de la dynamique. Ainsi, un choix adéquat de la sémantique de la dynamique est important pour obtenir un modèle correct.

Dans cette section, nous détaillons deux sémantiques de la dynamique des AN qui sont les plus répandues dans la littérature : celle qui considère une mise à jour des évolutions du systèmes purement *asynchrone* et une autre purement *synchrone*. Le choix d'une sémantique par rapport à une autre dépend principalement des systèmes complexes étudiés et des abstractions mathématiques choisies par le modélisateur.

Dynamique des AN à travers les transitions jouables :

Comme il a été expliqué formellement dans la définition 3.1 en page 56, un état global d'un AN est un n-uplet des états locaux actifs de tous les automates du AN en question, contenant exactement un seul état local pour chaque automate et $n \in \mathbb{N}$ est le nombre total de ses automates. Dans la suite, nous donnons des notations en lien avec l'introduction de la dynamique des AN et qui facilite la compréhension des définitions qui suivent.

Dans un AN, $(\Sigma, \mathcal{S}, \mathcal{T})$, l'état local actif d'un automate donné $a \in \Sigma$ dans un état global $\zeta \in \mathcal{S}$ est noté par $\zeta[a]$. Pour tout état local $a_i \in \mathbf{LS}$, nous notons aussi : $a_i \in \zeta$ si et seulement si $\zeta[a] = a_i$. Cela signifie que le composant a est dans son niveau discret étiqueté par i dans l'état global ζ.

Pour un ensemble donné des états locaux $X \subseteq \mathbf{LS}$, nous étendons cette notation à $X \subseteq \zeta$ si et seulement si $\forall a_i \in X$, $\zeta[a] = a_i$, ce qui signifie que tous les états locaux de X sont actifs dans ζ.

Dans la définition 3.2 ci-dessous, nous formalisons la caractéristique de la jouabilité d'une transition locale dans un état global. En effet, chaque transition locale du réseaux peut être jouable ou pas dans un état global, et si elle est *jouable*, alors elle peut être activée (ou pas selon la sémantique) pour faire évoluer l'automate qui la contient de son niveau actif vers un autre.

Une transition locale $a_i \xrightarrow{\ell} a_j$ est dite jouable dans un état global ζ, si et seulement si le niveau discret actif de a dans ζ est égal à i (i.e., $\zeta[a] = a_i$) et que tous les états locaux de sa condition, ℓ, sont aussi des états locaux actifs dans l'état global ζ (i.e., $\forall b_k \in \ell$, $\zeta[b] = b_k$). Ainsi, nous présentons formellement, dans la définition 3.2, l'ensemble de toutes les transitions locales qui sont jouables dans un état global du système.

Définition 3.2 (Transitions locales jouables)**.** Soit $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$ un réseau d'automates et $\zeta \in \mathcal{S}$ un état global de ce réseau. L'ensemble des transitions locales jouables dans ζ, noté par $\mathrm{J}(\zeta)$ est défini par :

$$\mathrm{J}(\zeta) = \{a_i \xrightarrow{\ell} a_j \in \mathcal{T} \mid \zeta[a] = a_i \wedge \forall b_k \in \ell,\ \zeta[b] = b_k\}.$$

Exemple. Dans le AN de l'exemple de la figure 3.2 en page 56, l'ensemble des transitions locales jouables dans l'état global $\zeta = \langle a_1, b_2, c_0, d_1\rangle \sim 1201$ est :

$\mathrm{J}(1201) = \{a_1 \xrightarrow{\{b_2\}} a_0, b_2 \xrightarrow{\{c_0\}} b_0, d_1 \xrightarrow{\{a_1\}} d_0\}$.

$\mathrm{J}(1201)$ est trouvé selon la définition 3.2 en parcourant toutes les transitions locales du réseau. En effet, par exemple, pour la transition locale $a_1 \xrightarrow{\{b_2\}} a_0$, $\zeta[a] = a_1$ et $\ell = \{b_2\}$ avec $\zeta[b] = b_2$.

Toutes les autres transitions locales de cet exemple, différentes de celles dans $\mathrm{J}(1201)$, ne sont pas jouables dans 1201. C'est parce que, pour chacune, l'origine ou au moins un des états locaux de sa condition n'est pas actif dans l'état global $\zeta = 1201$. Par exemple, la transition locale $c_0 \xrightarrow{\{a_1, b_0\}} c_1$ n'est pas jouable car $\zeta[b] = b_2 \neq b_0$ or b_0 est une condition nécessaire pour que cette transition locale soit jouable.

L'activation des transitions locales qui sont activées dépend principalement de la sémantique choisie. En effet, par exemple, en asynchrone pur, une transition locale qui est jouable n'est pas nécessairement activée.

Un chemin qui s'intègre dans la dynamique d'un AN correspond à l'activation successive des *transitions globales* entre des états globaux du réseau. Ces transitions globales sont le résultat de l'activation d'un ensemble de transitions locales.

Nous présentons dans la suite de cette section les deux sémantiques : asynchrone et synchrone. La différence entre elles est due principalement au fait que dans un même état global, l'ensemble des transitions locales activées selon une sémantique n'est pas nécessairement le même selon l'autre. Chaque ensemble représente ainsi une transition globale dans le graphe d'états.

Description des sémantiques asynchrone et synchrone :

Le choix de la sémantique (asynchrone ou synchrone) amène à avoir des ensembles différents des transitions locales activées et par la suite des différentes transitions globales. Ceci mène alors à avoir des évolutions dynamiques différentes.

Définition 3.3 (Sémantique Asynchrone)**.** Soit $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$ un réseau d'automates, $\zeta \in \mathcal{S}$ un état global de $\mathcal{AN}$ et $\mathsf{J}(\zeta)$ l'ensemble des transitions locales jouables dans ζ. L'ensemble des *transitions globales jouables* dans ζ selon une sémantique *asynchrone* est :

$$U^{\mathsf{asyn}}(\zeta) = \{\{a_i \xrightarrow{\ell} a_j\} \mid a_i \xrightarrow{\ell} a_j \in \mathsf{J}(\zeta)\}.$$

Exemple. Pour l'exemple de la figure 3.2 en page 56, l'ensemble des transitions locales jouables dans l'état global $\zeta = \langle a_1, b_2, c_0, d_1\rangle \sim 1201$ est : $\mathsf{J}(1201) = \{a_1 \xrightarrow{\{b_2\}} a_0, b_2 \xrightarrow{\{c_0\}} b_0, d_1 \xrightarrow{\{a_1\}} d_0\}$. Ainsi, l'ensemble des transitions globales jouables dans 1201 selon la sémantique asynchrone est $U^{\mathsf{asyn}}(1201)$, défini ci-dessous :

$U^{\mathsf{asyn}}(1201) = \{\{a_1 \xrightarrow{\{b_2\}} a_0\}, \{b_2 \xrightarrow{\{c_0\}} b_0\}, \{d_1 \xrightarrow{\{a_1\}} d_0\}\}$.

Une transition globale jouable dans un état global donné et qui a été trouvée selon la sémantique asynchrone (définition 3.3 ci-dessus) contient toujours une et une seule transition locale parmi celles qui sont jouables dans cet état. Ainsi, chaque transition globale activée dans une trajectoire trouvée selon la sémantique asynchrone ne change l'état local que d'un et un seul automate du réseau. Par conséquent, dans le graphe d'états, deux états globaux successifs ont exactement une différence au niveau d'un même automate. Autrement dit, il y a un seul composant qui a deux états locaux différents entre les deux états globaux successifs. Cette différence est due à l'activation de la transition locale contenue dans la transition globale activée dans le premier état global et qui le fait évoluer vers le deuxième état global.

Par contre, une transition globale jouable dans une sémantique synchrone (définition 3.4 ci-dessous) peut contenir plusieurs transitions locales. En effet, c'est un ensemble de toutes les transitions locales jouables dans cet état global.

Il est à noter qu'une transition globale trouvée par rapport à la sémantique synchrone dans un état global ζ, $U^{\mathsf{syn}}(\zeta)$, ne peut pas contenir plus qu'une transition locale jouable dans un automate : $\forall u \in U^{\mathsf{syn}}(\zeta)$ et $\forall a \in \Sigma$, $|u \cap \mathcal{T}_a| = 1$.

Définition 3.4 (Sémantique synchrone)**.** Soit $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$ un réseau d'automates et $\zeta \in \mathcal{S}$ un état global. L'ensemble des transitions globales jouables dans ζ selon une sémantique synchrone est :

$$U^{\mathsf{syn}}(\zeta) = \{u \subseteq \mathcal{T} \mid \forall a \in \Sigma, (\mathsf{J}(\zeta) \cap \mathcal{T}_a = \emptyset \Rightarrow u \cap \mathcal{T}_a = \emptyset) \wedge (\mathsf{J}(\zeta) \cap \mathcal{T}_a \neq \emptyset \Rightarrow |u \cap \mathcal{T}_a| = 1)\}.$$

Exemple. Pour le même exemple de la figure 3.2 en page 56, l'ensemble des transitions locales jouables dans l'état global $\zeta = \langle a_1, b_2, c_0, d_1 \rangle \sim 1201$ est (le même que celui en asynchrone) : $\mathrm{J}(1201) = \{a_1 \xrightarrow{\{b_2\}} a_0, b_2 \xrightarrow{\{c_0\}} b_0, d_1 \xrightarrow{\{a_1\}} d_0\}$.

Ainsi, l'ensemble des transitions globales jouables dans 1201 selon la sémantique synchrone est $U^{\mathsf{syn}}(1201)$, défini ci-dessous :

$U^{\mathsf{syn}}(1201) = \{\{a_1 \xrightarrow{\{b_2\}} a_0, b_2 \xrightarrow{\{c_0\}} b_0, d_1 \xrightarrow{\{a_1\}} d_0\}\}$.

Dans la sémantique synchrone, toutes les transitions qui sont jouables doivent être activées tout en satisfaisant la contrainte qui dit qu'une et une seule transition locale est activée par automate. Cette contrainte est déduite du fait que les transitions locales ne doivent pas être en *concurrence* pour qu'elles puissent s'activer au même temps. Nous expliquons davantage ce point de concurrence entre les transitions locales d'un même automate dans la suite.

Une fois que la sémantique est choisie, il est possible d'identifier l'évolution de la dynamique d'un AN. Autrement dit, calculer son graphe d'états de transitions. Nous notons que c'est l'activation des transitions locales contenues dans les transitions globales jouables qui mène à l'évolution de la dynamique du réseau. En effet, chaque transition locale jouable, une fois activée, change l'état de l'automate qui la contient. Par exemple, si le réseau est dans l'état 1201 et qu'on active la transition locale $\tau = a_1 \xrightarrow{\{b_2\}} a_0$ (dans une sémantique asynchrone), alors dans l'état suivant, l'état local de a sera égal à $\mathsf{dest}(\tau) = a_0$ et l'état global du système sera 0201.

Dans la suite, quand il n'y a pas d'ambiguïtés et quand les résultats sont applicables sur les deux sémantiques, nous notons par U la sémantique choisie parmi U^{asyn} et U^{syn}. La définition 3.5 ci-dessous formalise la notion d'une transition globale dans un graphe d'états de transitions quelle que soit la sémantique U (qui peut être même une sémantique combinée entre l'asynchrone et le synchrone). Elle montre aussi que pour toutes les transitions locales τ qui appartiennent à une transition globale u, son changement présenté par $\mathsf{dest}(\tau)$ doit apparaître dans l'état suivant.

Nous donnons avant de définir les transitions globales quelques notations qui sont en lien avec la définition de la dynamique avec les transitions globales des AN.

Pour tout état local $a_i \in \mathbf{LS}$, $\zeta \Cap a_i$ représente un état global qui est identique à ζ, à l'exception de l'état local de a qui a été remplacé par a_i :

$$(\zeta \Cap a_i)[a] = a_i \wedge \forall b \in \Sigma \setminus \{a\}, (\zeta \Cap a_i)[b] = \zeta[b].$$

Nous généralisons cette notation par l'ensemble des états locaux $X \subseteq \mathbf{LS}$ ayant au plus un état local par automate, alors, $\forall a \in \Sigma, |X \cap \mathcal{S}_a| \leq 1$ où $|S|$ est le nombre d'éléments dans un ensemble S. Dans ce cas, $\zeta \Cap X$ est l'état global ζ avec chaque état local pour chaque automate qui a été remplacé par l'état local du même automate dans X, s'il existe :

$$\forall a \in \Sigma, (X \cap \mathcal{S}_a = \{a_i\} \Rightarrow (\zeta \Cap X)[a] = a_i) \wedge (X \cap \mathcal{S}_a = \emptyset \Rightarrow (\zeta \Cap X)[a] = \zeta[a]).$$

Définition 3.5 (Transition globale)**.** Soit $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$ un réseau d'automates, $\zeta_1, \zeta_2 \in \mathcal{S}$ deux états globaux et U une sémantique (i.e., $U \in \{U^{\text{asyn}}, U^{\text{syn}}\}$). L'évolution de la dynamique du $\mathcal{AN}$ d'un état global ζ_1 vers un autre ζ_2 est assurée par la relation entre ces deux états. Cette relation est appelée une transition globale trouvée selon une sémantique U, notée par $\zeta_1 \rightarrow_U \zeta_2$, et définie par :

$$\zeta_1 \rightarrow_U \zeta_2 \Longleftrightarrow \exists u \in U(\zeta_1) \text{ tel que } \zeta_2 = \zeta_1 \Cap \{\mathsf{dest}(\tau) \in \mathbf{LS} \mid \tau \in u\}.$$

L'état ζ_2 est appelé le *successeur* de ζ_1, et l'état ζ_1 est appelé le *prédécesseur* de ζ_2.

Exemple. Les figures 3.3 et 3.4 illustrent des parties des graphes d'états de transitions trouvés selon respectivement les sémantiques asynchrone et synchrone du modèle de la figure 3.2 en page 56. Chaque transition globale est représentée par une flèche entre deux états globaux successifs.

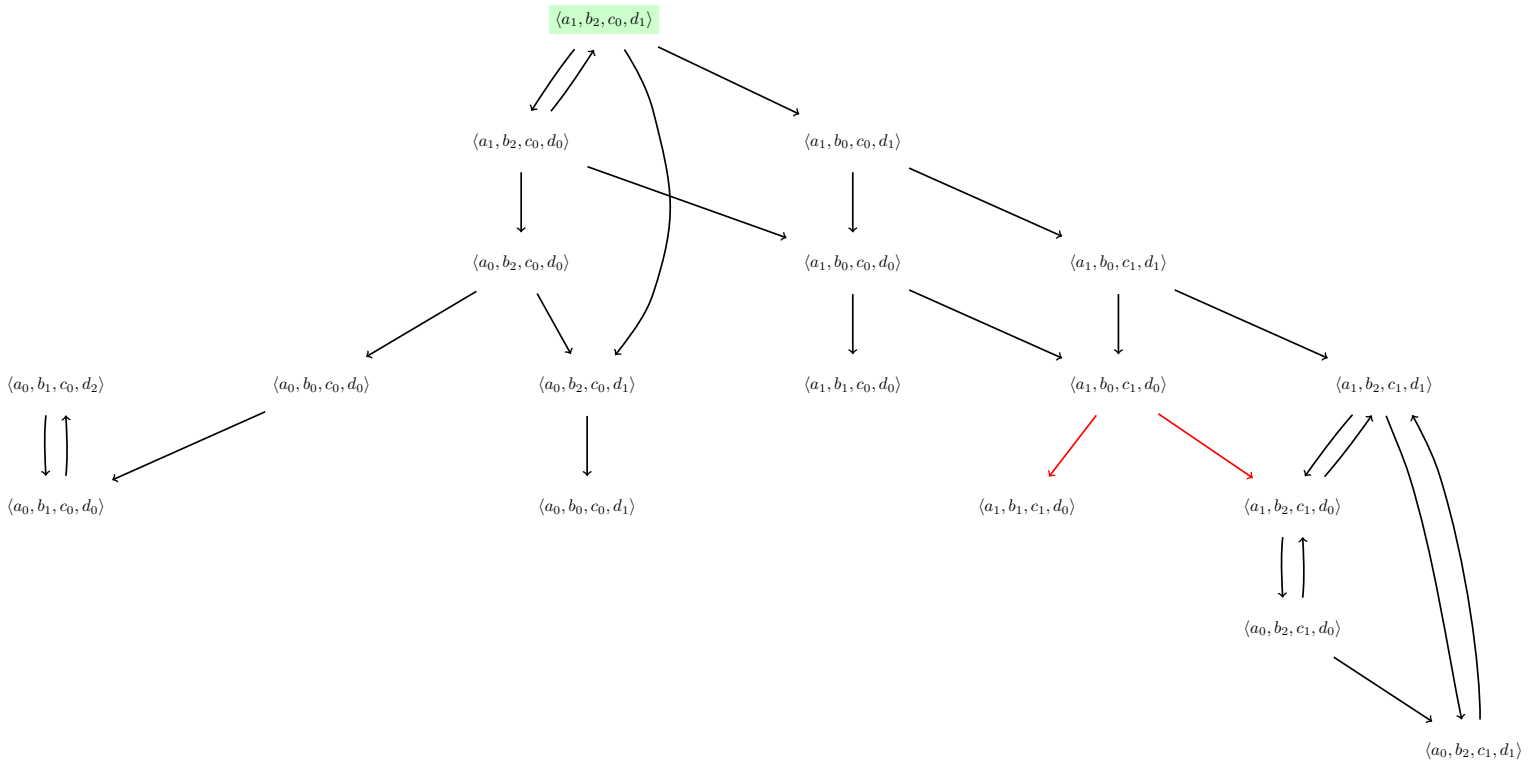


Figure 3.3 : Une partie du graphe d'états de transitions du modèle des réseaux d'automates donné dans la figure 3.2 en page 56 avec une sémantique **asynchrone**.

Par exemple, on retrouve l'état global $\zeta = \langle a_1, b_2, c_0, d_1\rangle \sim 1201$ (coloré en vert) dont les transitions globales jouables dans les sémantiques asynchrone et synchrones sont calculées dans les exemples précédents en pages 59 et 60. On retrouve qu'en asynchrone $\langle a_1, b_2, c_0, d_1\rangle$ a 3 successeurs car il a 3 transitions globales jouables ; $U^{\text{asyn}}(1201) = \{\{a_1 \xrightarrow{\{b_2\}} a_0\}, \{b_2 \xrightarrow{\{c_0\}} b_0\}, \{d_1 \xrightarrow{\{a_1\}} d_0\}\}$. Alors qu'en synchrone, il n'a qu'une seule transition globale jouable, $U^{\text{syn}}(1201) = \{\{a_1 \xrightarrow{\{b_2\}} a_0, b_2 \xrightarrow{\{c_0\}} b_0, d_1 \xrightarrow{\{a_1\}} d_0\}\}$ ce qui justifie le fait qu'il n'a qu'un seul successeur.

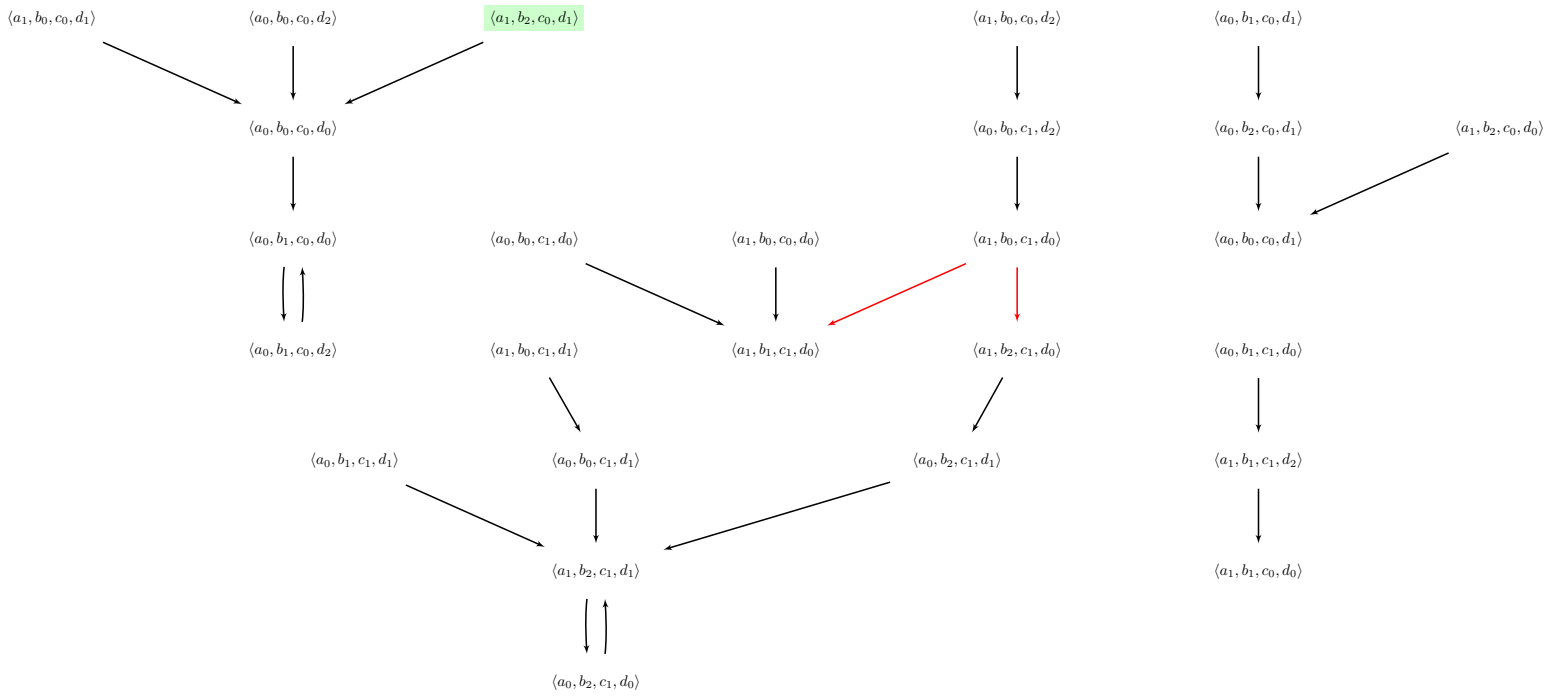


Figure 3.4 : Une partie du graphe d'états de transitions du modèle des réseaux d'automates donné dans 3.2 en page 56 avec une sémantique **synchrone**.

Il est remarquable que dans le graphe d'états trouvé selon la sémantique asynchrone (figure 3.3), la plupart des états ont plus qu'un successeur. Ceci crée ainsi un non-déterminisme dans l'évolution dynamique du réseau. Par contre, selon la sémantique synchrone (figure 3.4) ces mêmes états ont un seul successeur. Ceci à l'exception de l'état $\langle a_1, b_0, c_1, d_0\rangle$ où les flèches sortantes sont colorées en rouge dans les deux figures. En effet, la sémantique synchrone présente une évolution dynamique déterministe sauf dans le cas où il y a une concurrence entre les transitions locales d'un même automate (voir ci-après).

Selon Harvey et Bossomaier (Harvey & Bossomaier, 1997), les systèmes asynchrones sont biologiquement plus plausibles pour plusieurs phénomènes que les système synchrones. En effet, le comportement synchrone global observé dans la nature provient généralement simplement du comportement asynchrone local. Il peut être considéré comme la représentation du cas où plusieurs événements se produisent assez près dans le temps afin de ne pas considérer des états intermédiaires. En revanche, lorsque les réseaux de régulation biologique synchrones ont été étudiés (Kauffman, 1969), peu a été fait sur la contrepartie asynchrone (Thomas, 1991), bien qu'il y ait des preuves que la plupart des systèmes vivants sont réglés par une sémantique combinée, synchrone et asynchrone. Par exemple, dans (Noual & Sené, 2017), les auteurs ont prouvé que la considération qu'un ensemble de transitions puissent s'activer en parallèle, et d'autres pas, pourrait avoir un impact significatif sur le comportement asymptotique du réseau.

Ainsi, nous présentons dans la section 3.3 suivante, un comportement combiné (synchrone et asynchrone) de la dynamique des réseaux d'automates avec le temps. Nous montrons comment l'intégration des délais dans les transitions locales a permis de raffiner la dynamique des AN en définissant les cas précis auxquels il est possible d'autoriser un parallélisme entre des transitions et quelles sont les transitions qui ne peuvent pas s'activer

ensemble et celles qui peuvent.

Concurrence entre les transitions locales du même automate :

Dans un état global donné, si deux transitions locales différentes sont jouables et peuvent changer l'état du même automate vers des niveaux différents, alors elles ne peuvent pas être activées en parallèle. Donc pour chaque automate a, une et une seule transition locale est choisie parmi ses $\mathcal{T}_a$ pour être activée dans une transition globale selon la sémantique synchrone. Ceci est important à savoir car c'est la seule cause d'un non-déterminisme qui peut apparaitre dans la dynamique des AN dans la sémantique synchrone (voir les flèches en rouge dans la figure 3.4 en page 62).

En effet, deux transitions locales sont dites concurrentes dans un état global ζ, si elles appartiennent au même automate a, telles qu'elles sont toutes les deux jouables dans ζ, et qu'elles font évoluer a vers deux états locaux distincts. Une telle conséquence apparaît quand les deux transitions locales ont des destinations différentes mais le même origine tout en ayant des conditions compatibles ; c'est-à-dire tous les états locaux des automates dans leurs conditions peuvent être actifs dans un même état global ζ.

Dans la définition 3.6 ci-dessous, nous définissons formellement les transitions locales concurrentes.

Définition 3.6 (Transitions locales concurrentes)**.** Soit $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$ un réseau d'automates, $a \in \Sigma$ est un automate et $\tau_1, \tau_2 \in \mathcal{T}_a$ deux transitions locales qui font évoluer cet automate. τ_1 et τ_2 sont en *concurrence* si et seulement si elles ont des destinations différentes et s'il existe un état global dans lequel les deux sont jouables :

$$\exists \zeta \in \mathcal{S} \text{ telles que } \tau_1 \in \mathsf{J}(\zeta) \wedge \tau_2 \in \mathsf{J}(\zeta) \wedge \mathsf{dest}(\tau_1) \neq \mathsf{dest}(\tau_2).$$

Exemple. Une concurrence existe dans l'état global $\zeta = \langle a_1, b_0, c_1, d_0 \rangle \sim 1010$ entre les transitions locales $b_0 \xrightarrow{\{d_0\}} b_1$ et $b_0 \xrightarrow{\{a_1, c_1\}} b_2$. En effet, ces deux transitions locales qui appartiennent au même automate, sont toutes les deux jouables dans l'état 1010. En revanche, elles ne peuvent pas être activées au même temps. car elles changent l'état local du même automate, b_0, vers deux états locaux différents, b_1 et b_2. Ainsi, dans la sémantique synchrone ces deux transitions locales ne doivent pas appartenir à la même transition globale : $\mathsf{J}(1010) = \{b_0 \xrightarrow{\{d_0\}} b_1, b_0 \xrightarrow{\{a_1, c_1\}} b_2\}$.

Par conséquent, il y a deux ensembles de transitions globales différents dans 1010 selon la sémantique synchrone : $U^{\mathsf{syn}}(1010) = \{\{b_0 \xrightarrow{\{d_0\}} b_1\}, \{b_0 \xrightarrow{\{a_1, c_1\}} b_2\}\}$.

Pour cet état global 1010, on peut déduire que l'ensemble des transitions globales jouables dans la sémantique asynchrone et dans la sémantique synchrone est le même. En effet, il n'y a pas d'autres transitions locales jouables dans $\mathsf{J}(1010)$. Donc $U^{\mathsf{asyn}}(1010) = U^{\mathsf{syn}}(1010)$.

On retrouve dans les figures 3.3 et 3.4 en pages 61 et 62, l'état global $\zeta = \langle a_1, b_0, c_1, d_0 \rangle \sim 1010$ duquel sortent deux flèches rouges pointant vers ses successeurs qui ont été calculés par l'activation de chaque transition globale trouvée par les deux ensembles $U^{\mathsf{asyn}}(1010)$ et $U^{\mathsf{syn}}(1010)$. Comme il est prévu, les successeurs de cet état sont les mêmes dans les deux figures, puisque $U^{\mathsf{asyn}}(1010) = U^{\mathsf{syn}}(1010)$.

Les transitions locales concurrentes produisent ainsi un non-déterminisme à l'intérieur de l'automate qui les contient et qui cause le seul non-déterminisme dans la dynamique globale

du réseau dans la sémantique synchrone (voir figure 3.4 en page 62). Une telle conséquence apparaît quand les deux transitions locales ont la même origine et des conditions compatibles, mais ayant des destinations qui sont différentes. Nous étudions plus profondément la dynamique des AN et le non déterminisme dans ses graphes d'états par rapport aux sémantiques asynchrone et synchrone dans le chapitre 5. Nous y développons aussi les définitions des propriétés dynamiques des AN.

Dans la suite de ce chapitre, nous introduisons l'enrichissement du formalisme des AN par l'intégration d'une composante temporelle : un délai pour chaque transition locale. Nous montrons aussi comment cette incorporation d'une telle information temporelle permet le raffinement de la dynamique des systèmes biologiques.

3.3 Intégration du temps dans les réseaux d'automates

Il existe des systèmes biologiques où la variable du temps a un rôle important dans leurs dynamiques. Cette variable qui représente le temps nécessaire pour que le système permute d'un état global vers un autre, varie selon les transitions qui sont activées. Comme par exemple, dans le système de l'horloge circadienne : la durée de temps nécessaire pour que le basculement d'un état vers un autre soit effectif constitue une caractéristique importante du modèle. Nous étudions davantage le modèle de l'horloge circadienne au chapitre 4.

Présenter ces systèmes avec le formalisme des AN, que nous avons introduits dans la section précédente, ne fournit pas un cadre de modélisation qui permette d'avoir de l'information *chronométrique* à propos du système modélisé. En effet, un graphe d'états trouvé pour un AN donné n'illustre que la chronologie entre ces états. Par exemple, dans les graphes d'états des figures 3.3 et 3.4 en pages respectives 61 et 62, il n'y a que des trajectoires de la dynamique du système et pas d'information sur le temps mis par chaque transition pour s'activer.

Ainsi, nous proposons de raffiner la dynamique des AN par l'incorporation d'une variable de temps dans le modèle. Elle représente la durée d'activation d'une transition locale. Comme il a été déjà mentionné, on note par T-AN, le nom de l'extension du formalisme des AN (définition 3.7 ci-dessous). Dans le formalisme des T-AN, chaque transition locale a une durée de temps spécifique durant laquelle elle se produit appelée le "*délai*". Ce délai est un entier positif noté par δ ($\delta \in \mathbb{N}$). Dans un T-AN, les transitions locales sont alors appelées des "*transitions locales temporisées*".

Nous présentons dans cette section, la sémantique de la dynamique des T-AN. Cette sémantique adapte la dynamique des AN à cet enrichissement avec les délais dans les transitions locales. En effet, nous montrons que des conflits pourront se créer entre les transitions locales temporisées à cause des ressources partagées (éléments nécessaires pour qu'une transition soit jouable). La sémantique que nous proposons permet non seulement d'adapter la dynamique du modèle à ce conflit, mais aussi de raffiner cette dynamique par rapport à celle des AN par la création de nouveaux comportements ou encore par la suppression d'autres.

3.3.1 Définitions des réseaux d'automates avec le temps (T-AN)

Ce formalisme est différent des AN par les transitions locales temporisées $a_i \xrightarrow[\delta]{\ell} a_j$. On note par δ le délai propre pour chaque transition locale temporisée. Il représente le temps nécessaire pour qu'une transition locale s'active. En effet, au cours de la modélisation des phénomènes de régulation, le délai permet de capturer la période requise entre l'ordre d'activation pour la production d'une protéine et la synthèse effective de cette dernière ainsi que la synthèse des produits résultants. Nous définissons formellement ci-dessous un T-AN dont les délais sont représentés sous forme d'entiers ($\delta \in \mathbb{N}$).

Définition 3.7 (Réseau d'automates avec le temps (T-AN))**.** *Un réseau d'automates avec le temps* est un triplet $(\Sigma, \mathcal{S}, \mathcal{T})$ tels que :

- $\Sigma = \{a, b, \ldots\}$ est l'ensemble fini des identifiants des automates ;
- Pour tout $a \in \Sigma$, $\mathcal{S}(a) = \{a_i, \ldots, a_j\}$ est l'ensemble fini des états locaux de l'automate a ; $\mathcal{S} = \prod_{a\in\Sigma} \mathcal{S}(a)$ est l'ensemble fini des états globaux ;
  $\mathbf{LS} = \cup_{a\in\Sigma}\mathcal{S}(a)$ est l'ensemble de tous les états locaux.
- Pour tout $a \in \Sigma$, $\mathcal{T}_a = \{a_i \xrightarrow[\delta]{\ell} a_j \in \mathcal{S}_a \times \wp(\mathbf{LS} \setminus \mathcal{S}_a) \times \mathbb{N} \times \mathcal{S}_a \mid a_i \neq a_j\}$ est l'ensemble des *transitions locales temporisées* de l'automate a ; $\mathcal{T} = \bigcup_{a\in\Sigma} \mathcal{T}_a$ est l'ensemble de toutes les transitions locales temporisées dans le modèle.

Nous notons $\tau = a_i \xrightarrow[\delta]{\ell} a_j \in \mathcal{T}_a \Leftrightarrow \tau \in \mathcal{T}$ avec $\mathcal{T}_a \subset \mathcal{T}$. On note aussi, $\mathsf{ori}(\tau) = a_i$, $\mathsf{dest}(\tau) = a_j$, $\mathsf{cond}(\tau) = \ell$ et $\mathsf{delai}(\tau) = \delta$, tels que $\ell \in \wp(\mathbf{LS} \setminus \mathcal{S}_a)$ et $\delta \in \mathbb{N}$.

Nous notons qu'un AN peut être traduit en T-AN en considérant que toutes ses transitions locales ont un délai qui est égal à 1 (voir propriété 3.1 ci-dessous). En effet, les délais dans un T-AN sont des entiers. De plus, dans un état global et à un instant t, si une transition locale est activée, alors le changement apparaît dans l'état global successeur à l'instant $t + 1$.

Propriété 3.1. Si $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$ est un AN, alors il pourrait être considéré comme un T-AN : $\mathcal{AN}' = (\Sigma, \mathcal{S}, \mathcal{T}')$ avec $\forall\tau \in \mathcal{T}, \exists\tau' \in \mathcal{T}'$ tels que $\mathsf{ori}(\tau') = \mathsf{ori}(\tau)$, $\mathsf{dest}(\tau') = \mathsf{dest}(\tau)$, $\mathsf{cond}(\tau') = \mathsf{cond}(\tau)$ et $\mathsf{delai}(\tau') = 1$.

Par conséquent, selon la propriété 3.1 ci-dessus, la définition 3.2 en page 58, qui définit l'ensemble des transitions locales jouables dans un état global d'un AN, est alors aussi valable pour les transitions locales temporisées dans un T-AN.

Exemple. La figure 3.5 ci-dessous représente un T-AN $(\Sigma, \mathcal{S}, \mathcal{T})$ avec 3 automates : a, b et z. Comme le montre cette figure, tous les automates du modèle ont chacun 2 états locaux 0 et 1, et il contient 5 transitions locales temporisées. Nous réutilisons ce modèle dans d'autres endroits dans ce manuscrit de thèse, et nous l'appelons l'*"exemple jouet"*. Il est décrit de façon textuelle ci-dessous :

- $\Sigma = \{a, b, z\}$,
- $\mathcal{S}_a = \{a_0, a_1\}$, $\mathcal{S}_b = \{b_0, b_1\}$, $\mathcal{S}_z = \{z_0, z_1\}$,
- $\mathcal{T} = \{\tau_1 = a_0 \xrightarrow[4]{\emptyset} a_1,\ \tau_2 = a_1 \xrightarrow[1]{\{z_1\}} a_0,\qquad \tau_3 = b_1 \xrightarrow[3]{\{a_1\}} b_0,$
  $\tau_4 = z_0 \xrightarrow[3]{\{a_0, b_1\}} z_1,\ \tau_5 = z_1 \xrightarrow[2]{\{b_0\}} z_0,\ \}.$

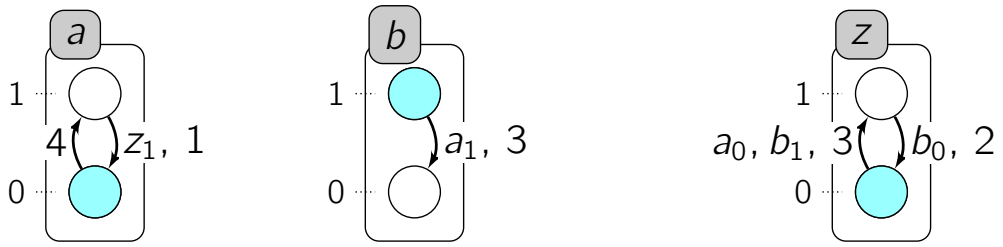


Figure 3.5 : Exemple d'un réseau d'automates avec le temps, appelé "exemple jouet".

L'ajout du délai comme un paramètre supplémentaire dans une transition locale, et prendre en considération le conflit entre les transitions, exigent une étude supplémentaire de la sémantique du modèle. En effet, des nouveaux comportements pourraient avoir lieu ou d'autres pourraient être supprimés. En plus, d'autres conflits pourraient être identifiés dans la dynamique des T-AN.

Nous étudions alors dans la suite de ce chapitre, la dynamique des T-AN, et nous expliquons quels sont les problèmes qui peuvent être rencontrés lors de la simulation d'un T-AN. Ensuite, nous défendons les hypothèses que nous adoptons pour les résoudre. Il est à noter que pour la sémantique de la dynamique des T-AN, nous considérons un comportement asynchrone pour chaque automate, d'une part, et un comportement synchrone dans le réseau global. En effet, dans l'approche que nous adoptons, nous permettons, sans aucune obligation, l'activation en parallèle des transitions locales temporisées, si et seulement il n'y a pas de *conflits* entre elles mais nous ne permettons pas l'activation des transitions locales appartenant à un même automate. Nous définissons dans la section suivante les transitions locales temporisées en conflit et pourquoi elles ne peuvent pas être activées en parallèle. Nous introduisons ensuite la sémantique de la dynamique générale des T-AN.

3.3.2 Fonctionnement des transitions locales temporisées

Quand nous parlons d'une simulation d'un modèle, nous parlons de la construction des chemins qui peuvent être suivis par la dynamique du modèle. Dans certains cas, lors de la simulation d'un T-AN, des conflits entre les transitions locales temporisées, qui n'existent pas lors de la simulation d'un AN, pourraient se créer. En effet, ces conflits sont principalement dûs au partage des ressources entre les transitions locales temporisées. Par exemple, durant la période d'activation d'une transition locale temporisée τ, il est possible qu'une autre transition locale temporisée τ' soit activée telle que τ' partage les mêmes ressources que τ (voir l'exemple suivant ci-dessous). Donc, il faut vérifier si τ et τ' peuvent s'activer en parallèle et si ce n'est pas le cas, et il faut voir quel effet ceci aurait sur l'évolution de la dynamique du modèle.

Quand il y a des conflits entre les transitions, on peut dire que la plupart du temps, les transitions les plus rapides (i.e., ayant un délai minimal) ont plus de chance de passer mais ceci pourrait être aussi une histoire de probabilité. En effet, comme il a été montré dans la thèse de Fitime (Fippo-Fitime, 2016), le délai intervient dans le calcul de la probabilité de l'activation d'une transition locale par rapport à une autre. En revanche, dans cette thèse, nous étudions tous les cas possibles, c'est-à-dire toutes les évolutions possibles de la dynamique d'un T-AN.

Ainsi, il est important d'avoir une sémantique bien définie qui puisse d'une part, gérer ce genre de conflits et d'autre part, offrir un raffinement de la dynamique tout en restant

exhaustive sur l'ensemble des possibilités. Et c'est ce que nous présentons dans cette section.

Pour ceci, nous introduisons d'abord quelques exemples des conflits qui peuvent exister entre les transitions locales temporisées. Puis, nous les définissons formellement dans la définition 3.8 en page 68.

Transitions locales temporisées en conflit :

Dans le reste du chapitre, et pour la clarté des descriptions, nous notons par **état**$(\mathcal{AN}, t)$, l'état global du T-AN $\mathcal{AN}$ à l'instant t.

Exemple. Soit $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$ un T-AN, tels que $\Sigma = \{a, b, c\}$ et $\mathcal{T} = \{\tau_1, \tau_2, \tau_3\}$ avec $\tau_1 = a_1 \xrightarrow[5]{\{b_0\}} a_0$, $\tau_2 = c_0 \xrightarrow[2]{\emptyset} c_1$ et $\tau_3 = b_0 \xrightarrow[2]{\{c_1\}} b_1$. Supposant qu'à $t = 0$, **état**$(\mathcal{AN}, 0) = \langle a_1, b_0, c_0\rangle$, ainsi τ_1 et τ_2 sont jouables à $t = 0$.

Ci-dessous, deux évolutions de la dynamique de ce modèle (ou deux chemins) illustrant le changement des niveaux locaux des automates pendant 5 unités de temps. Chaque transition globale entre deux états globaux successifs est étiquetée par les noms des transitions locales qui sont en cours :

t	0	1	2	3	4	5
	$\langle a_1, b_0, c_0\rangle \xrightarrow[\tau_1+\tau_2]{} \langle a_1, b_0, c_0\rangle$	$\xrightarrow[\tau_1+\tau_2]{} \langle a_1, b_0, \boldsymbol{c_1}\rangle$	$\xrightarrow{\tau_1} \langle a_1, b_0, c_1\rangle$	$\xrightarrow{\tau_1} \langle a_1, b_0, c_1\rangle$	$\xrightarrow{\tau_1} \langle \boldsymbol{a_0}, b_0, c_1\rangle$	
			$\xrightarrow{\tau_3} \langle a_1, b_0, c_1\rangle$	$\xrightarrow{\tau_3} \langle a_1, \boldsymbol{b_1}, c_1\rangle$		

On remarque qu'à $t = 2$, la transition locale temporisée τ_2 termine son activité et c change instantanément du niveau 0 au niveau 1. À ce moment, τ_3 est devenue jouable. En revanche, τ_3 entre en conflit avec τ_1 qui est en cours. En effet, τ_1 nécessite la présence de b_0 pendant 5 unités de temps, et à cet instant, i.e., à $t = 2$, il reste encore 3 unités de temps pour atteindre sa fin d'activation. Or, τ_3 cause le changement de b du niveau 0 vers le niveau 1. Ce changement est fait au bout d'une durée de temps qui est égale à 2 et qui est inférieure à 3 (tel que 3 est la période restante de τ_1 avant qu'elle se termine). Ainsi, si τ_3 est activée à $t = 2$, τ_1 ne peut plus aboutir à sa fin, parce qu'elle ne sera plus jouable à $t = 4$ (car le niveau local de b n'est plus égal à 0) et elle sera alors "interrompue" par τ_3. C'est pourquoi dans le deuxième chemin (à la deuxième ligne) a ne change pas de niveau.

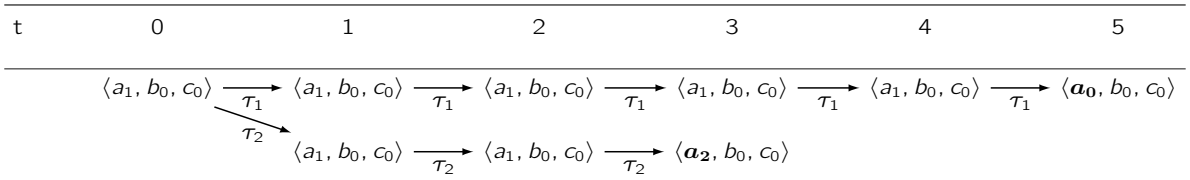
Ce conflit, comme le montre le graphe ci-dessus, crée un non-déterminisme qui est similaire à celui dans la sémantique asynchrone ; les transitions locales s'activent séparément. En effet, ce n'est qu'un choix *stochastique* fait par le système pour suivre l'un de ces deux chemins et qui l'emmènent vers deux états stables différents : $\langle a_0, b_0, c_1\rangle$ ou $\langle a_1, b_1, c_1\rangle$.

Il est à noter que si delai(τ_3) est strictement supérieur à 2, il n'y aurait pas eu de conflit entre τ_1 et τ_3. En effet, b ne change alors pas de niveau durant la période restante, c'est-à-dire entre $t = 2$ et $t = 5$. Et dans ce cas, à $t = 2$, τ_1 et τ_3 seront activées en parallèle $(\tau_1 + \tau_3)$. Par conséquent, à $t = 5$, il aurait eu les deux changements de a et de b et l'état global du système serait $\langle a_0, b_1, c_1\rangle$ et qui est un état stable (car aucune transition est jouable).

Ci-dessous, un autre exemple illustre un autre conflit sur les ressources partagées entre les transitions locales temporisées.

Exemple. Soit $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$ un T-AN tels que $\Sigma = \{a, b, c\}$ et $\tau_1, \tau_2, \tau_3 \in \mathcal{T}$ avec $\tau_1 = a_1 \xrightarrow[5]{\{b_0\}} a_0$ et $\tau_2 = a_1 \xrightarrow[3]{\{c_0\}} a_2$.

Ci-dessous, deux trajectoires (illustrées par une succession d'états globaux) de l'évolution dynamique de ce réseau pendant 5 unités de temps. Nous supposons que **état**$(\mathcal{AN}, 0) = \langle a_1, b_0, c_0\rangle$, donc les transitions locales τ_1 et τ_2 qui sont concurrentes (car elles changent le même automate vers des niveaux différents) sont toutes les deux jouables à $t = 0$.



Pour cet exemple, on remarque que le choix entre les transitions locales temporisées en conflit τ_1 et τ_2 est fait dès le début (c'est-à-dire à $t = 0$). En effet, ces deux transitions sont toutes les deux jouables à $t = 0$. Elles sont dites en conflit vue leur concurrence, parce que l'une, τ_2, a une influence positive sur a alors que l'autre, τ_1, l'inhibe.

Ce dernier exemple montre que la définition 3.6 en page 63 qui définit les transitions locales en concurrence dans un AN, est alors aussi valable pour les transitions locales temporisées dans un T-AN. Effectivement, le temps n'intervient pas ici et cette concurrence existe toujours que ce soit dans les AN ou dans les T-AN. En effet, les transitions locales temporisées qui appartiennent au même automate et qui sont toutes les deux jouables dans un même état global mais qui font changer l'état local de l'automate vers deux niveaux différents sont nécessairement en conflit.

Nous formalisons ainsi dans la définition 3.8 ci-dessous les cas où nous considérons que deux transitions locales temporisées sont en conflit.

Définition 3.8 (Transitions locales temporisées en conflit)**.** Soit $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$ un T-AN et $a, b \in \Sigma$ avec $a \neq b$. Soit $\tau' \in \mathcal{T}_a$ est activée à un instant t' et $\tau \in \mathcal{T}_b$ est devenue jouable à un instant t tel que $t' < t$.
τ est *en conflit* avec τ' si et seulement si :

$$\begin{gathered}(\mathsf{ori}(\tau) = \mathsf{ori}(\tau')) \\ \vee\ \mathsf{ori}(\tau) \in \mathsf{cond}(\tau')\ \wedge\ t + \mathsf{delai}(\tau) < t' + \mathsf{delai}(\tau') \\ \vee\ (\mathsf{ori}(\tau') \in \mathsf{cond}(\tau) \wedge t + \mathsf{delai}(\tau) > t' + \mathsf{delai}(\tau')).\end{gathered}$$

Dans la définition 3.8 ci-dessus, le premier cas stipule que lorsqu'une transition τ est en conflit avec une autre τ' à un instant t, c'est quand $\mathsf{ori}(\tau) = \mathsf{ori}(\tau')$. Ainsi, dans un chemin, si une transition τ' qui modifie un automate a est en cours, alors, aucune autre transition ne peut être activée pour modifier le même automate a.

Exemple. Soit τ une transition locale temporisée, telle que $\tau = a_i \xrightarrow[\delta]{\ell} a_j$ a été activée à un instant t. Ce qui implique que, $t + \delta$ est l'instant auquel τ se termine et $t' + \delta'$ est l'instant auquel une autre transition $\tau' = b_k \xrightarrow[\delta']{\ell'} b_h$ se termine avec t' est l'instant auquel τ' est activée. Nous considérons le cas où $t' < t$.

La figure 3.6 ci-dessous représente la durée d'activation de τ en rouge et la durée d'activation de τ' en vert.

Selon la définition 3.8, $\tau = a_i \xrightarrow[\delta]{\ell} a_j$ est en conflit avec τ' si a_i est une ressource nécessaire à τ' ; $a_i \in \ell'$ (i.e., $\mathsf{ori}(\tau) \in \mathsf{cond}(\tau')$) et si τ se termine avant τ', autrement dit, $t' + \delta' > t + \delta$ (i.e., $t + \mathsf{delai}(\tau) > t' + \mathsf{delai}(\tau')$). Ceci est illustré dans le 1er cas de la figure 3.6 ci-dessous.

Ainsi, a_i (i.e., $\mathsf{ori}(\tau)$) ne sera plus disponible pour participer à l'activation de la transition τ' parce qu'il est modifié par τ à l'instant $t + \delta$ qui est inférieur à $t' + \delta'$. Donc, il existe un conflit entre ces deux transitions et elles ne peuvent pas être activées en parallèle.

Le 2ème cas de la figure 3.6 ci-dessous, illustre un autre conflit qui pourrait exister. La transition τ est en conflit avec τ' si dans sa condition, elle nécessite une ressource qui est en train d'être changée par τ' (i.e., $\mathsf{ori}(\tau') \in \mathsf{cond}(\tau)$) et si ce changement se termine avant l'écoulement de toute la période nécessaire pour que τ se termine ; c'est-à-dire avant $t + \delta$, autrement dit, si $t + \delta > t' + \delta'$ ($t + \mathsf{delai}(\tau) > t' + \mathsf{delai}(\tau')$).

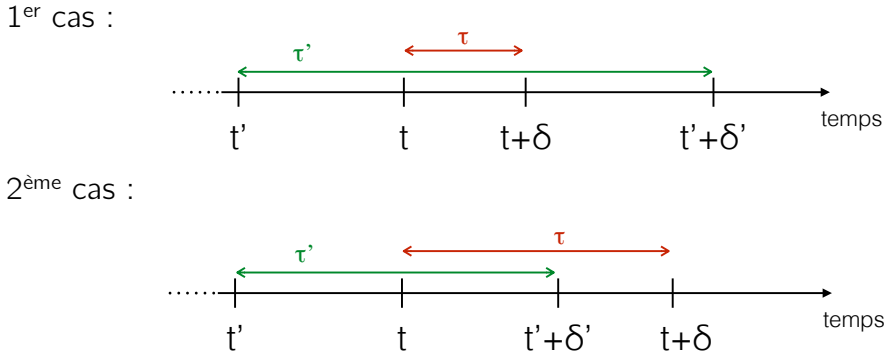


Figure 3.6 : Exemple d'un chevauchement entre deux transitions pendant leurs activations et qui pourrait mener à un conflit en cas de partage de ressources.

Pour gérer les conflits qui peuvent exister entre les transitons locales temporisées, nous supposons alors des hypothèses qui sont similaires à celles évoquées dans (Goldstein & Bockmayr, 2013). En effet, nous considérons que les transitions locales temporisées qui sont en conflit *se bloquent* les unes les autres. Ainsi, dans la suite, nous développons la sémantique de la dynamique des T-AN par rapport à ce blocage.

Blocage entre les transitions locales temporisées :

Les transitions locales temporisées qui sont en cours d'activation jouent un rôle important dans l'évolution de la dynamique d'un T-AN. En effet, elles permettent d'identifier les composants qui sont en cours de changement ou qui participent à un changement en cours. Ainsi, nous considérons qu'à chaque instant t, le réseau est non seulement caractérisé par son état global mais aussi par son ensemble de transitions locales temporisées qui sont en cours (noté $\mathsf{C}(t)$). Autrement dit, $\mathsf{C}(t)$ est un ensemble de transitions locales temporisées qui ont été activées à des instants antérieurs (i.e., avant l'instant t) mais qui ne sont pas encore terminées.

Définition 3.9 (Transitions locales temporisées en cours)**.** Soient $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$ un T-AN, $\mathsf{C}(t)$ l'ensemble des paires $\mathcal{T} \times \mathbb{N}$ est *l'ensemble des transitions locales temporisées en cours* à l'instant t et qui ont été activées à des instants antérieurs t', $t' < t$, avec $t, t' \in \mathbb{N}$:

$$\mathsf{C}(t) := \{(\tau', t') \in \mathcal{T} \times \mathbb{N} \mid \mathsf{delai}(\tau') = \delta' \wedge t < t' + \delta'\}.$$

Exemple. Dans la première trajectoire du premier exemple en page 67 (première ligne dans le tableau), à l'instant $t = 1$, les transitions locales temporisées qui sont en cours sont τ_1 et τ_2. Donc, $\mathsf{C}(1) = \{(\tau_1, 0), (\tau_2, 0)\}$ avec 0 est l'instant auquel τ_1 et τ_2 ont été activées. Ensuite, à $t = 2$, τ_2 se termine ; en effet l'automate c change de c_0 vers c_1. Ainsi, il n'y a que τ_1 qui est en cours. Donc, $\mathsf{C}(2) = \{(\tau_1, 0)\}$. Pour l'instant final, $t = 5$, auquel τ_1 se termine, il n'y a aucune transition qui est en cours, ainsi, $\mathsf{C}(5) = \emptyset$.

De même pour la deuxième trajectoire, on trouve qu'à l'instant $t = 2$, $\mathsf{C}(2) = \{(\tau_2, 0)\}$ et à l'instant $t = 3$, $\mathsf{C}(3) = \emptyset$.

Dans la suite, nous supposons que pour tout chemin, à l'instant initial, $t = 0$, $\mathsf{C}(0) = \emptyset$.

La définition 3.10 ci-dessous donne une formalisation des transitions locales temporisées bloquées par les transitions qui sont en cours et qui sont en conflit avec elles (définition 3.8 en page 68).

Définition 3.10 (Transitions locales temporisées bloquées)**.** Soient $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$ un T-AN, $t \in \mathbb{N}$ et $\mathsf{C}(t) \in \mathcal{T} \times \mathbb{N}$ est l'ensemble des transitions locales temporisées en cours à l'instant t. *L'ensemble des transitions locales temporisées bloquées* de $\mathcal{AN}$ par $\mathsf{C}(t)$ à l'instant t est défini par $\mathsf{B}(\mathcal{AN}, \mathsf{C}(t), t)$ ainsi :

$$\mathsf{B}(\mathcal{AN}, \mathsf{C}(t), t) := \{\tau \in \mathcal{T} \mid \exists \tau' \in \mathsf{C}(t) \text{ telle que } \tau \text{ et } \tau' \text{ sont en conflit}\}.$$

Dans notre sémantique, nous donnons la priorité à la transition qui est en cours. Ainsi, pendant leurs activités, les transitions locales temporisées qui sont en cours bloquent l'activité de celles qui deviennent jouables et qui sont en conflit avec elles.

Exemple. Dans l'exemple précédent, en page 68, τ est devenue jouable quand τ' est en cours. Puisque τ est en conflit avec τ', alors τ est bloquée par τ'. La période de blocage est entre le moment où τ devient jouable, i.e., à l'instant t, et l'instant auquel τ' termine son activité, i.e., à l'instant $t' + \delta'$ (voir figure 3.6 en page 69). Autrement dit, le blocage commence quand τ (transition bloquée) est devenu jouable et finit quand τ' (transition en cours et qui la bloque) se termine.

Propriété 3.1. *Soit τ une transition locale temporisée jouable à un instant t telle qu'elle est en conflit avec une autre τ' en cours ; $(\tau', t') \in \mathsf{C}(t)$ et $\mathsf{delai}(\tau') = \delta'$.*
Ainsi, τ est bloquée pendant la période définie par l'intervalle de temps : $[t, t' + \delta'[$.
Sachant que dans notre sémantique, nous ne manipulons que des entiers, alors on peut dire que cet intervalle est équivalent à $[t, t' + \delta' - 1]$ si $\delta' \neq 1$.

La propriété 3.1 est complètement cohérente avec la sémantique de la dynamique des T-AN que nous adoptons et défendons dans ce chapitre : il n'est pas possible d'activer une transition locale temporisée en conflit avec une autre qui est en cours. Ce qui implique que l'ensemble des transitions *activées* à un instant donné, dépend de l'ensemble des transitions qui sont en cours et plus précisément de celles qui les bloquent. En effet, si une transition τ est bloquée par τ' à un instant t, alors elle ne sera débloquée que quand τ' se termine. Autrement dit, l'instant du déblocage de τ est le même que celui auquel τ' se termine (i.e., $t' + \delta'$ tel que $\mathsf{delai}(\tau') = \delta'$ et t' l'instant auquel τ' a été activée).

Cependant, nous permettons aux ressources de τ de participer à l'activation d'autres transitions. C'est-à-dire, si par exemple $\tau = a_i \xrightarrow[\delta]{\ell} a_j$ et $b_k \in \ell$, alors b_k peut faire partie des conditions ($\mathsf{cond}(\tau'')$) d'autres transitions actives (τ'').

De plus, nous n'interdisons pas à $\mathsf{ori}(\tau)$ d'appartenir à des conditions d'autres transitions τ'' qui sont actives ; $\mathsf{ori}(\tau) \in \mathsf{cond}(\tau'')$. En revanche, il ne faut pas que l'instant auquel τ se termine soit supérieur à l'instant auquel τ'' se termine (autrement dit, τ et τ'' ne sont pas en conflit). En effet, sinon a_i ne serait pas disponible pour toute la période d'activité de τ''.

La définition 3.11 étend la notion des transitions locales jouables dans un AN (définition 3.2 en page 58) en des transitions locales temporisées *franchissables*. En effet, pour qu'une transition locale temporisée soit franchissable, il faut qu'elle soit jouable et en plus non bloquée. Ainsi, pour qu'une transition locale temporisée soit activée à un instant t, il ne suffit pas qu'elle soit jouable dans l'état global du réseau à cet instant, il faut qu'elle ne soit pas bloquée. De ce fait, il faut prendre aussi en considération les conflits qui peuvent exister entre une transition jouable et les transitions en cours appartenant à $\mathsf{C}(t)$.

Définition 3.11 (Transitions locales temporisées franchissables)**.** Soient $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$ un T-AN et $\zeta = \textbf{état}(\mathcal{AN}, t)$ tels que $\zeta \in \mathcal{S}$ et $t \in \mathbb{N}$. Soit $\mathsf{J}(\zeta)$ l'ensemble des transitions locales jouables dans ζ (définition 3.2 en page 58).
$\mathsf{C}(t) \in \mathcal{T} \times \mathbb{N}$ l'ensemble des transitions locales temporisées en cours à l'instant t et $\mathsf{B}(\mathsf{C}(t), t)$ l'ensemble des transitions locales temporisées bloquées par $\mathsf{C}(t)$ à l'instant t.
L'ensemble des transitions locales temporisées franchissables dans ζ par rapport à $\mathsf{C}(t)$ à t est défini par :

$$\mathsf{FL}(\zeta, \mathsf{C}(t), t) := \{a_i \xrightarrow[\delta]{\ell} a_j \in \mathsf{J}(\zeta) \setminus \mathsf{B}(\mathsf{C}(t), t)\}.$$

Exemple. Soient $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$ un T-AN et $\zeta = \textbf{état}(\mathcal{AN}, t)$ tels que $\zeta \in \mathcal{S}$ et $t \in \mathbb{N}$.

Soit $\mathsf{J}(\zeta) = \{\tau_1, \tau_2, \tau_3, \tau_4, \tau_5, \tau_6, \tau_7\}$ l'ensemble des transitions locales temporisées jouables dans ζ. Supposant que $\mathsf{C}(t) = \{(\tau_1, t_1), (\tau_2, t_2), (\tau_3, t_3)\}$ est l'ensemble des transitions locales temporisées en cours à t telles que chaque τ_i a été activée à l'instant t_i, avec $i = \{1, 2, 3\}$.

On a aussi $\mathsf{B}(\mathsf{C}(t), t) = \{\tau_1, \tau_2, \tau_3, \tau_6, \tau_8, \tau_9\}$ est l'ensemble des transitions locales temporisées bloquées par les transitions de $\mathsf{C}(t)$ à l'instant t.

Ainsi, $\mathsf{FL}(\zeta, \mathsf{C}(t), t) = \mathsf{J}(\zeta) \setminus \mathsf{B}(\mathsf{C}(t), t) = \{\tau_4, \tau_5, \tau_7\}$ est l'ensemble des transitions locales temporisées franchissables à l'instant t.

Pour éviter toute ambiguïté de compréhension, nous appelons parfois, en dehors des définitions, les ensembles par leurs noms sans les paramètres entre les parenthèses. En occurrence, FL=FL$(\zeta, \mathsf{C}(t), t)$ et B=B$(\mathsf{C}(t), t)$.

Propriété 3.2. *Toute transition locale temporisée jouable peut être franchissable ou bloquée.*

$$\tau \in \mathsf{J} \Rightarrow \tau \in \mathsf{FL} \oplus \tau \in \mathsf{B}.$$

On a alors :

$$\mathsf{B} \cap \mathsf{FL} = \emptyset.$$

Ainsi, toute transition bloquée n'est pas franchissable et toute transition franchissable n'est pas bloquée :

$$\tau \in \mathsf{J} \textit{ telle que } \tau \in \mathsf{B} \Rightarrow \tau \notin \mathsf{FL}.$$

$$\tau \in \mathsf{J} \textit{ telle que } \tau \in \mathsf{FL} \Rightarrow \tau \notin \mathsf{B}.$$

Dans les T-AN, les transitions locales temporisées activées à un instant donné ne sont plus trouvées qu'à partir des transitions locales jouables dans l'état global du réseau à cet instant (comme c'est le cas dans la dynamique des AN sans les délais, section 3.2.1 en page 55).

Dans la sémantique que nous présentons pour la dynamique des T-AN, la transition globale activée dans un état global du réseau, notée AG, n'est qu'un des éléments de l'ensemble des transitions globales franchissables noté FG : AG $\in$ FG (définition 3.12 ci-dessous). On peut ainsi dire que FG est l'ensemble des transitions globales candidates à un instant donné parmi lesquels AG est sélectionnée. Par conséquent, trouver FG à chaque instant et dans un état global du chemin, permet de trouver la transition globale qui pourrait être activée AG.

Nous notons que chaque élément de l'ensemble FG, c'est-à-dire chaque transition globale franchissable est un ensemble de transitions locales temporisées franchissables en parallèle noté FL (définition 3.11 en page 71). Autrement dit, c'est un ensemble de transitions locales temporisées qui peuvent être activées simultanément sans avoir des conflits entre elles.

Définition 3.12 (L'ensemble des transitions globales franchissables)**.** Soient $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$ un T-AN et **état**$(\mathcal{AN}, t) = \zeta$ tels que $\zeta \in \mathcal{S}$ et $t \in \mathbb{N}$. $\mathsf{C}(t) \in \mathcal{T} \times \mathbb{N}$ l'ensemble des transitions locales temporisées en cours à l'instant t et $\mathsf{FL}(\mathcal{AN}, \zeta, \mathsf{C}(t), t)$ l'ensemble des transitions locales temporisées franchissables de $\mathcal{AN}$ dans ζ par rapport à $\mathsf{C}(t)$ à l'instant t.
FG est *l'ensemble des transitions globales franchissables* tel que chaque élément est un ensemble de transitions locales temporisées franchissables dans ζ par rapport à $\mathsf{C}(t)$ à l'instant t :

$$\mathsf{FG}(\zeta, \mathsf{C}(t), t) := \{\mathsf{AG} \subseteq \mathsf{FL}(\zeta, \mathsf{C}(t), t) \mid$$
$$\forall \tau \in \mathsf{AG}, \nexists \tau' \in \mathsf{AG} \text{ telle que } \tau' \in \mathsf{B}(\mathsf{AG} \setminus \{\tau'\}, t)\}.$$

Exemple. Reprenons le même exemple de la page 71. Si nous récapitulons, sachant que **état**$(\mathcal{AN}, t) = \zeta$, nous avons :

$\mathsf{J}(\zeta) = \{\tau_1, \tau_2, \tau_3, \tau_4, \tau_5, \tau_6, \tau_7\}$, $\mathsf{C}(t) = \{(\tau_1, t_1), (\tau_2, t_2), (\tau_3, t_3)\}$,
$\mathsf{B}(\mathsf{C}(t), t) = \{\tau_1, \tau_2, \tau_3, \tau_6, \tau_8, \tau_9\}$, et $\mathsf{FL}(\zeta, \mathsf{C}(t), t) = \{\tau_4, \tau_5, \tau_7\}$.

Supposant que τ_4 et τ_7 sont en conflit donc elles ne peuvent pas être activées en parallèle, alors FG, l'ensemble des transitions globales franchissables, est égal à :

$\mathsf{FG}(\zeta, \mathsf{C}(t), t) = \{\{\tau_4, \tau_5\}, \{\tau_5, \tau_7\}\}$.

Par conséquent, puisque AG est la transition globale activée telle que AG $\in$ FG alors elle peut être égale à l'un des éléments de FG : $\mathsf{AG} = \{\tau_4, \tau_5\}$ ou $\mathsf{AG} = \{\tau_5, \tau_7\}$.

Ainsi, on peut voir que FG est un ensemble des ensembles de transitions locales temporisées franchissables. Autrement dit, chaque élément de FG est un ensemble de transitions locales temporisées franchissables. Donc, chaque élément de FG est un sous-ensemble de FL (voir propriété 3.3 ci-dessous).

Propriété 3.3. *Chaque transition globale activée à un instant donné notée* AG *est un élément de* FG*, et elle est incluse dans* FL *:*

$$\mathsf{AG} \subseteq \mathsf{FL}.$$

Nous présentons un récapitulatif de la distribution des transitions locales temporisées d'un T-AN à un instant donné dans la figure 3.7 ci-dessous.

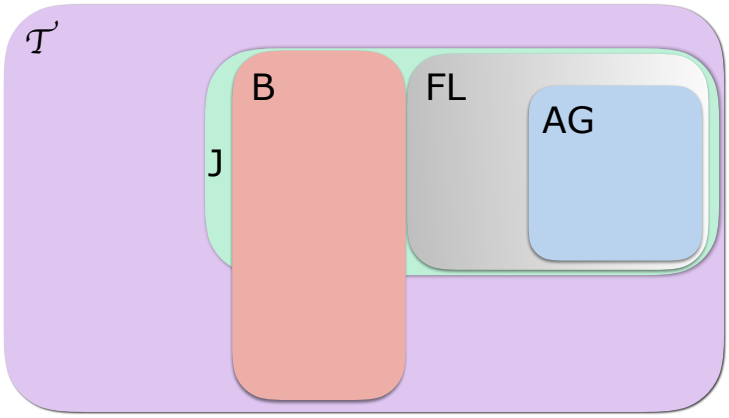


Figure 3.7 : La distribution des transitions locales temporisées $\mathcal{T}$ à un instant t selon les définitions 3.7–3.12.

Nous supposons dans la figure 3.7, que toutes les transitions locales temporisées du réseau, $\mathcal{T}$, sont représentées par l'ensemble violet, et celles qui sont jouables dans l'état global du réseau à un instant t, J, sont représentées par l'ensemble vert ($J \subset \mathcal{T}$). Les transitions locales temporisées qui sont bloquées, B, sont représentées par l'ensemble rouge, et celles qui sont franchissables, FL, sont représentées par l'ensemble gris.

On rappelle que si une transition locale temporisée est jouable et non bloquée alors elle est franchissable (voir propriété 3.2 en page 71). Donc les transitions de J sont partagées en deux sous-ensembles distincts qui sont B et FL tels que $FL \cap B = \emptyset$. En effet, chaque transition de J est nécessairement incluse dans l'un de ces deux ensembles : elle est franchissable ou (exclusif) elle est bloquée.

Finalement, AG, représentée par l'ensemble en bleu, est la transition globale activée à l'instant t et qui contient l'ensemble des transitions locales temporisées activées. Ces dernières sont sélectionnées parmi celles qui sont franchissables dans FL car $AG \subseteq FL$ (voir propriété 3.3 ci-dessus). En effet, pas toutes les transitions locales qui sont franchissables sont activées, parce qu'il pourrait y avoir un conflit entre elles, donc elles ne peuvent pas être activées en parallèle (c'est le cas des transitions τ_4 et τ_7 dans exemple précédent en page 72).

Activité des transitions locales temporisées :

Nous présentons ci-après comment la dynamique générale d'un T-AN est établie. La définition 3.13 ci-dessous montre comment à chaque instant t, l'ensemble des transitions locales temporisées qui sont *actives* (noté $A(t)$) est calculé. Cet ensemble contient d'une part, les transitions locales temporisées qui ont été activées à des instants antérieurs mais qui ne se terminent pas à t ; c'est l'ensemble des transitions en cours $C(t)$ (définition 3.9 en page 69) et d'autre part, il contient les transitions locales temporisées qui viennent d'être activées à l'instant t. Ces dernières appartiennent toutes à la transition globale activée à t : AG. On rappelle que AG est sélectionnée parmi les transitions globales franchissables FG.

Nous supposons que dans l'état initial du chemin (i.e., à $t = 0$), aucune transition n'est bloquée : $B = \emptyset$. En effet, ceci est cohérent avec l'hypothèse qu'à $t = 0$, aucune transition

n'est en cours : $\mathsf{C}(0) = \emptyset$. Ainsi toutes les transitions locales temporisées qui sont jouables sont aussi franchissables (définition 3.11 en page 71).

Définition 3.13 (Transitions locales temporisées actives)**.** Soient $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$ un T-AN et **état**$(\mathcal{AN}, t) = \zeta$ tels que $\zeta \in \mathcal{S}$ et $t \in \mathbb{N}$. Soit $\mathsf{C}(t) \in \mathcal{T} \times \mathbb{N}$ l'ensemble des transitions locales temporisées en cours à l'instant t et $\mathsf{FG}(\zeta, \mathsf{C}(t), t)$ l'ensemble des transitions globales franchissables.
L'ensemble des *transitions locales temporisées actives* de $\mathcal{AN}$ à l'instant t est :

$$\mathsf{A}(t) := \begin{cases} \{(\tau, 0) \mid \tau \in \mathsf{AG} \text{ et } \mathsf{AG} \in \mathsf{FG}(\zeta, \emptyset, t)\} & \text{si } t = 0 \\ \{(\tau, t) \mid \tau \in \mathsf{AG} \text{ et } \mathsf{AG} \in \mathsf{FG}(\zeta, \mathsf{C}(t), t) \bigcup \mathsf{C}(t)\} & \text{si } t > 0. \end{cases}$$

Dans le but de clarifier la différence entre la période de temps pendant laquelle une transition locale temporisée est en cours (définition 3.9 en page 69) et la période pendant la quelle elle est active (définition 3.13 ci-dessus), nous proposons la figure 3.8 ci-dessous.

En fait, il est important de faire la différence entre les deux périodes, car la période pendant laquelle une transition locale temporisée τ pourrait bloquer une autre est celle pendant laquelle τ est en cours ; i.e., quand $\tau \in \mathsf{C}$. Comme le montre la figure 3.8 ci-dessous (par l'intervalle en rouge ouvert des deux cotés), la période où τ est en cours n'inclut pas les bornes de l'intervalle. La borne inférieure n'est que l'instant où τ est devenue franchissable et commence son activité (en vert). Et la borne supérieure indique l'instant où τ termine son activité (en violet).

D'autre part, la figure 3.8 montre par l'intervalle en bleu, la période de temps pendant laquelle τ est active. Cet intervalle est semi-ouvert en borne supérieure. En effet, on considère qu'à l'instant où τ se termine, τ n'est plus active.

Exemple. La figure 3.8 ci-dessous, différencie les intervalles de temps et les instants importants pour l'étude de l'activité d'une transition et en l'occurrence ici de τ_σ. À l'instant t_1, τ_σ est franchissable (en vert). Ainsi, à partir de cet instant elle est active (en bleu) ; dans l'intervalle de temps semi-ouvert $[t_1, t_1 + t_\sigma[$ avec $\mathsf{delai}(\tau_\sigma) = t_\sigma$. Son activité se termine à l'instant $t_1 + t_\sigma$ (en violet). Finalement, elle est en cours (en rouge) dans l'intervalle de temps ouvert $]t_1, t_1 + t_\sigma[$. On a encore, si $t_\sigma \neq 1$ et sachant que 1 est la durée minimale qui sépare deux instants successifs (car les instants sont des entiers), alors τ_σ est en cours dans l'intervalle de temps fermé $[t_1 + 1, t_1 + t_\sigma - 1]$ (voir propriété 3.1 en page 70).

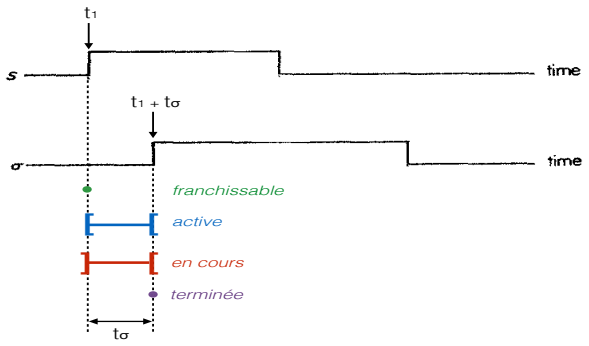


Figure 3.8 : La différence entre les intervalles de temps pendant l'activité de la transition locale temporisée $\tau_\sigma = \sigma_0 \xrightarrow[\tau_\sigma]{S_1} \sigma_1$. τ_σ est active dans l'intervalle de temps semi-ouvert à droite (en bleu). Elle est en cours dans l'intervalle de temps ouvert des deux bornes (en rouge). Les points en vert et en violet indiquent respectivement les instants auxquels τ_σ est franchissable et terminée. L'évolution de l'expression de S et σ est tirée de (Thomas, 1978).

Si $\mathsf{delai}(\tau) = 1$, alors on peut dire que la transition τ est d'une durée élémentaire, car nous ne manipulons que des entiers dans la sémantique de la dynamique des T-AN. Ainsi, τ n'est jamais considérée en cours dans notre calcul de la dynamique. En effet, si elle est franchissable à un instant t, l'intervalle pendant lequel elle est en cours est égal à $]t, t+1[$. Alors que, quand nous vérifions les instants pour lesquels une transition locale temporisée est en cours, nous trouvons que l'intervalle $]t, t+1[$ ne contient aucune valeur entière. Par exemple, le cas de la transition τ_1 dans le tableau 3.1 en page 76. τ_1 est activée à l'instant $t = 3$ ($\tau_1 \in \mathsf{A}$) et elle n'est pas considérée en cours à $t + 1 = 4$ ($\tau_1 \notin \mathsf{C}$) car son délai est égal à 1 ($\mathsf{delai}(\tau_1) = 1$). Nous expliquons davantage cet exemple dans la suite.

La définitions 3.14 ci-dessous définit formellement l'ensemble des transitions locales temporisées qui ont été activées antérieurement et qui se terminent à l'instant t.

Définition 3.14 (Transitions locales temporisées terminées)**.** Soient $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$ un T-AN et $\mathbf{état}(\mathcal{AN}, t) = \zeta$ tels que $\zeta \in \mathcal{S}$ et $t \in \mathbb{N}$. L'*ensemble des transitions locales temporisées terminées* à l'instant t est défini par :

$$\mathsf{AT}(t) := \{\tau' \in \mathcal{T} \mid (\tau', t') \in \mathsf{A}(t-1) \text{ tels que } t' + \delta' = t \text{ et } \mathsf{delai}(\tau') = \delta'\}.$$

Exemple. Nous donnons dans le tableau 3.1 ci-dessous un exemple d'un chemin du T-AN de l'exemple jouet de la figure 3.5 en page 66, à partir de l'état global initial $\langle a_0, b_1, z_0\rangle$.

On trouve qu'à chaque instant t, l'état global du réseau est indiqué dans la colonne de ζ avec $\mathbf{état}(\mathcal{AN}, t) = \zeta$. AT est l'ensemble des transitions qui sont terminées à t. Puis, B est l'ensemble des transitions locales temporisées bloquées par celles qui sont en cours C. FL représente l'ensemble des transitions locales temporisées franchissables dans ζ et à t : $\mathsf{FL} = \mathsf{FL}(\zeta, \mathsf{C}(t), t)$. $\mathsf{FG} = \mathsf{FG}(\zeta, \mathsf{C}(t), t)$ est l'ensemble des transitions globales franchissables à t. AG est sélectionnée à partir des éléments de FG et qui contient l'ensemble des transitions locales temporisées activées à t. Finalement, A est l'ensemble des transitions locales actives à t : $\mathsf{A} = \mathsf{A}(\mathcal{AN}, t)$. Tous ces ensembles sont calculés comme il a été indiqué précédemment dans les définitions 3.9–3.14.

t	ζ	AT	C	B	FL	FG	AG	A
0	$\langle a_0, b_1, z_0\rangle$	$\emptyset$	$\emptyset$	$\emptyset$	$\{\tau_1, \tau_4\}$	$\{\{\emptyset\}, \{\tau_1\}, \{\tau_4\}, \{\tau_1, \tau_4\}\}$	$\{\tau_1, \tau_4\}$	$\{(\boldsymbol{\tau_1, 0}), (\boldsymbol{\tau_4, 0})\}$
1	$\langle a_0, b_1, z_0\rangle$	$\emptyset$	$\{(\tau_1, 0), (\tau_4, 0)\}$	$\emptyset$	$\emptyset$	$\{\emptyset\}$	$\emptyset$	$\{(\tau_1, 0), (\tau_4, 0)\}$
3	$\langle a_0, b_1, \boldsymbol{z_1}\rangle$	$\{\tau_4\}$	$\{(\tau_1, 0)\}$	$\emptyset$	$\emptyset$	$\{\{\emptyset\}\}$	$\emptyset$	$\{(\tau_1, 0)\}$
4	$\langle \boldsymbol{a_1}, b_1, z_1\rangle$	$\{\tau_1\}$	$\emptyset$	$\emptyset$	$\{\tau_2, \tau_3\}$	$\{\{\emptyset\}, \{\tau_2\}, \{\tau_3\}\}$	$\{\tau_2\}$	$\{(\boldsymbol{\tau_2, 4})\}$
5	$\langle \boldsymbol{a_0}, b_1, z_1\rangle$	$\{\tau_2\}$	$\emptyset$	$\emptyset$	$\{\tau_1\}$	$\{\{\emptyset\}, \{\tau_1\}\}$	$\{\tau_1\}$	$\{(\boldsymbol{\tau_1, 5})\}$
6	$\langle a_0, b_1, z_1\rangle$	$\emptyset$	$\{(\tau_1, 5)\}$	$\emptyset$	$\emptyset$	$\{\{\emptyset\}\}$	$\emptyset$	$\{(\tau_1, 5)\}$
9	$\langle \boldsymbol{a_1}, b_1, z_1\rangle$	$\{\tau_1\}$	$\emptyset$	$\emptyset$	$\{\tau_2, \tau_3\}$	$\{\{\emptyset\}, \{\tau_2\}, \{\tau_3\}\}$	$\{\tau_3\}$	$\{(\boldsymbol{\tau_3, 9})\}$
10	$\langle a_1, b_1, z_1\rangle$	$\emptyset$	$\{(\tau_3, 9)\}$	$\{\tau_2\}$	$\emptyset$	$\{\{\emptyset\}\}$	$\emptyset$	$\{(\tau_3, 9)\}$
11	$\langle a_1, b_1, z_1\rangle$	$\emptyset$	$\{(\tau_3, 9)\}$	$\{\tau_2\}$	$\emptyset$	$\{\{\emptyset\}\}$	$\emptyset$	$\{(\tau_3, 9)\}$
12	$\langle a_1, \boldsymbol{b_0}, z_1\rangle$	$\{\tau_3\}$	$\emptyset$	$\emptyset$	$\{\tau_2, \tau_5\}$	$\{\{\emptyset\}, \{\tau_2\}, \{\tau_5\}, \{\tau_2, \tau_5\}\}$	$\{\tau_2, \tau_5\}$	$\{(\boldsymbol{\tau_2, 12}), (\boldsymbol{\tau_5, 12})\}$
13	$\langle \boldsymbol{a_0}, b_0, z_1\rangle$	$\{\tau_2\}$	$\{(\tau_5, 12)\}$	$\emptyset$	$\{\tau_1\}$	$\{\{\emptyset\}, \{\tau_1\}\}$	$\{\tau_1\}$	$\{(\tau_5, 12), (\boldsymbol{\tau_1, 13})\}$
14	$\langle a_0, b_0, \boldsymbol{z_0}\rangle$	$\{\tau_5\}$	$\{(\tau_1, 13)\}$	$\emptyset$	$\emptyset$	$\{\{\emptyset\}\}$	$\emptyset$	$\{(\tau_1, 13)\}$
17	$\langle \boldsymbol{a_1}, b_0, z_0\rangle$	$\{\tau_1\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\{\{\emptyset\}\}$	$\emptyset$	$\emptyset$

Table 3.1 : Un exemple d'un chemin du modèle T-AN de l'exemple jouet, le T-AN de la figure 3.5 en page 66, à partir de l'état initial $\langle a_0, b_1, z_0\rangle$ (à $t = 0$) jusqu'à l'atteinte de l'état stable $\langle a_1, b_0, z_0\rangle$ (à $t = 17$).

Ce tableau montre quelques conflits apparus entre des transitions locales temporisées et qui se sont alors bloquées mutuellement. Aux instants $t = 4$ et $t = 9$ l'état global du système est le même : $\textbf{état}(\mathcal{AN}, 4) = \textbf{état}(\mathcal{AN}, 9) = \langle a_1, b_1, z_1\rangle$, et les deux transitions locales temporisées τ_2 et τ_3 sont franchissables ($\mathsf{FL} = \{\tau_2, \tau_3\}$).

Il est à noter que ces deux transitions $\tau_2 = a_1 \xrightarrow[1]{z_1} a_0$ et $\tau_3 = b_1 \xrightarrow[3]{a_1} b_0$ sont en conflit dans $\langle a_1, b_1, z_1\rangle$. En effet, pour activer τ_3, il faut que l'automate a soit au niveau 1 (a_1) pendant au moins 3 unités de temps car $\mathsf{delai}(\tau_3) = 3$. Sinon, τ_3 ne pourra jamais aboutir à sa fin et changer b. Par contre, τ_2 cause le changement de a au bout d'une unité de temps ($\mathsf{delai}(\tau_1) = 1$). Ainsi, τ_2 et τ_3 ne peuvent pas appartenir à une même transition globale franchissable. Ce qui est le cas, dans le chemin du tableau 3.1, aux instants $t = 4$ et $t = 9$ où $\mathsf{FL} = \{\tau_2, \tau_3\}$ mais $\{\tau_2, \tau_3\} \notin \mathsf{FG}$. En effet, les deux transitions τ_2 et τ_3 ne peuvent qu'être activées séparément. Et ce qui justifie les éléments de FG ; $\mathsf{FG} = \{\{\emptyset\}, \{\tau_2\}, \{\tau_3\}\}$. En plus, une fois qu'une transition globale est choisie, notée AG, de l'ensemble des transitions globales franchissables FG, ($\mathsf{AG} \in \mathsf{FG}$), une des transitions τ_2 ou τ_3, sera bloquée jusqu'à ce que l'autre se termine.

Par exemple, à $t = 9$, la transition globale qui est activée est $\mathsf{AG} = \{\tau_3\}$. Ainsi, τ_2 est bloquée ($\mathsf{B} = \{\tau_2\}$) pendant la période de l'intervalle de temps : $]9, 9 + \mathsf{delai}(\tau_3)[$. Puisque nous ne manipulons que des entiers, cet intervalle est équivalent à $[9 + 1, 9 + \mathsf{delai}(\tau_3) - 1]$ (voir propriété 3.1 en page 70). On a $\mathsf{delai}(\tau_3) = 3$, alors τ_2 est bloquée dans l'intervalle de temps : $[10, 11]$ (voir tableau ci-dessus). Par conséquent, à $t = 12$, τ_2 est débloquée car τ_3 se termine ($\mathsf{AT} = \{\tau_3\}$). Cette fin est marquée par le changement de l'état local de b (de b_1 vers b_0).

Par contre, à $t = 4$, c'est τ_2 qui est activée : $\mathsf{AG} = \{\tau_2\}$. Ainsi τ_3 ne sera débloquée que quand τ_2 se termine, c'est-à-dire à l'instant qui est égal à $4 + \mathsf{delai}(\tau_2) = 4 + 1 = 5$. Autrement dit, τ_3 est bloquée pendant la période de temps définie par l'intervalle $]4, 5[$. Puisque dans notre sémantique nous ne manipulons que des entiers, et $]4, 5[$ ne contient pas des entiers, alors τ_3 ne sera pas considérée comme bloquée à aucun instant dans le chemin. Ainsi, elle n'est pas bloquée à l'instant suivant $t + 1 = 5$. Ce qui justifie que dans le chemin présentée dans le tableau ci-dessus, à $t = 5$, aucune transition n'est bloquée ($\mathsf{B} = \emptyset$). On en conclut qu'à $t = 4$, le conflit entre τ_2 et τ_3 n'a pas causé de blocage,

mais il exige quand même qu'elles ne soient pas activées simultanément. Donc, il y a un non-déterminisme dans la dynamique du T-AN à partir de cet état global (activer τ_2 ou activer τ_3).

On peut remarquer que ce dernier cas montre une dynamique similaire à un AN (sans les délais) selon une sémantique asynchrone (définition 3.3 en page 59). En effet, en asynchrone pur toutes les transitions locales temporisées ne peuvent pas être activées en parallèle, pourtant elles sont complètement indépendantes (i.e., il n'y a pas de conflit entre elles). Et c'est ce qui est énoncé dans la propriété 3.1 en page 65 : un T-AN où toutes les transitions locales temporisées ont un délai égal à 1, est assimilé à un AN.

Nous notons ainsi que la sémantique de la dynamique des T-AN est différente de celle trouvée en asynchrone pur. En effet, pour la sémantique de la dynamique des T-AN, on n'exige pas que toutes les transitions locales soient activées séparément, mais uniquement celles qui sont en conflit. D'autre part, la sémantique de la dynamique des T-AN est aussi différente du synchrone pur. En effet, il n'est pas possible d'activer toutes les transitions locales, quelques soient leurs activités, en parallèle. Comme il a été vu, il y a un choix qui est effectué selon des contraintes précises et qui permettent de vérifier si un ensemble de transitions locales puissent s'activer en parallèle ou non. Ce choix ne dépend pas seulement de l'état global actuel du réseau mais aussi de sa dynamique dans des états antérieurs.

Il est à noter qu'à chaque instant t, l'ensemble des transitons en cours $\mathsf{C}(t)$ est déduit de l'ensemble des transitions qui sont actives à l'instant précédent, c'est-à-dire à $t-1$: $\mathsf{A}(t-1)$. En effet, si une transition τ est active à $t-1$, et qu'elle ne se termine pas à l'instant t, alors elle est toujours active à t. Ainsi, $\mathsf{C}(t)$ est un sous-ensemble de $\mathsf{A}(t-1)$ (voir propriété 3.4 ci-dessous).

Propriété 3.4. *Soient $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$ un T-AN, $\mathsf{C}(t) \in \mathcal{T} \times \mathbb{N}$ l'ensemble des transitions locales temporisées en cours à l'instant t et $\mathsf{A}(t)$ l'ensemble des* transitions locales temporisées actives *à l'instant t. On a alors :*

$$\forall t > 0, \quad \mathsf{C}(t) \subseteq \mathsf{A}(t-1)$$

Exemple. Dans le tableau 3.1 en page 76, on peut remarquer que pour toutes les étapes de la trajectoire, de $t = 1$ à $t = 17$, $\mathsf{C}(t) \subseteq \mathsf{A}(t-1)$.

On remarque aussi, dans le tableau 3.1 en page 76, que ces deux ensembles $\mathsf{C}(t)$ et $\mathsf{A}(t-1)$ sont égaux s'il n'y a pas de changement d'état d'un automate à l'instant t. Autrement dit, s'il n'y a pas de transitions locales temporisées qui se terminent à t. Ce qui signifie que $\textbf{état}(\mathcal{AN}, t) = \textbf{état}(\mathcal{AN}, t-1)$. La propriété 3.5 ci-dessous formalise ce résultat.

Propriété 3.5. *Soient $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$ un T-AN, $\mathsf{C}(t) \in \mathcal{T} \times \mathbb{N}$ l'ensemble des transitions locales temporisées en cours à l'instant t et $\mathsf{A}(t-1)$ l'ensemble des* transitions locales temporisées actives *à l'instant $t-1$.*
On a alors :

$$\forall t > 0, \quad \mathsf{C}(t) = \mathsf{A}(t-1) \qquad \textit{ssi} \qquad \textbf{état}(\mathcal{AN}, t) = \textbf{état}(\mathcal{AN}, t-1).$$

Exemple. Dans le tableau 3.1 en page 76, on remarque qu'aux instants t auquel $\mathsf{AT}(t)=\emptyset$, $\mathsf{C}(t) = \mathsf{A}(t-1)$. En fait, ces instants sont : $t = 1$, $t = 6$ et $t = 10$. En effet, à ces instants,

on ne note pas des changements dans les états globaux du système par rapport à ses états globaux aux étapes précédentes, qui sont respectivement $t = 0$, $t = 5$ et $t = 9$. Alors, on a,
$\textbf{état}(\mathcal{AN}, 1) = \textbf{état}(\mathcal{AN}, 0)$, $\textbf{état}(\mathcal{AN}, 6) = \textbf{état}(\mathcal{AN}, 5)$ et $\textbf{état}(\mathcal{AN}, 10) = \textbf{état}(\mathcal{AN}, 9)$.

Ainsi, on peut vérifier qu'à ces instants $\mathsf{C}(t) = \mathsf{A}(t-1)$:
$\mathsf{C}(1) = \mathsf{A}(0)$, $\mathsf{C}(6) = \mathsf{A}(5)$ et $\mathsf{C}(10) = \mathsf{A}(9)$.

Dynamique globale des T-AN :

En se basant sur toutes les définitions de cette section, nous récapitulons dans la définition 3.15 ci-dessous, la sémantique générale selon laquelle les transitions locales temporisées se déroulent dans la dynamique d'un T-AN.

Rappelons qu'en simulant, à chaque instant, le T-AN a un seul état global dans lequel chaque automate n'a qu'un seul état local actif. L'ensemble de tous les états globaux du réseau est référencé par $\mathcal{S}$ avec $\mathcal{S} = \prod_{a \in \Sigma} \mathcal{S}(a)$. Pour un état global $\zeta \in \mathcal{S}$, $\zeta[a]$ est l'état local de l'automate a dans ζ. Nous notons aussi : $a_i \in \zeta \Leftrightarrow \zeta[a] = a_i$.

Définition 3.15 (La sémantique des réseaux d'automates avec le temps)**.** Soient $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$ un T-AN, et $\zeta_t = \textbf{état}(\mathcal{AN}, t)$ tels que $\zeta_t \in \mathcal{S}$ et $t \in \mathbb{N}$.
Soit $\mathsf{FG}(\zeta, \mathsf{C}(t), t)$ l'ensemble des transitions globales franchissables à l'instant t par rapport aux transitions locales en cours $\mathsf{C}(t)$. La transition globale activée à t est définie par : $\mathsf{AG} := \{\tau \in \mathcal{T}\}$ tel que $\mathsf{AG} \in \mathsf{FG}(\zeta, \mathsf{C}(t), t)$.
Alors,

$$\forall (\tau = a_i \xrightarrow[\delta]{\ell} a_j) \in \mathsf{AG} \implies \zeta_{t+\delta}[a] = a_j \text{ avec } \zeta_{t+\delta} = \textbf{état}(\mathcal{AN}, t+\delta).$$

L'état global successeur de $\mathcal{AN}$ à $t+1$: $\zeta_{t+1} = \textbf{état}(\mathcal{AN}, t+1)$ est défini par rapport à $\mathsf{AT}(t+1)$ qui est l'ensemble des transitions locales temporisées qui se terminent à $t+1$:

$\forall b \in \Sigma$, si $\zeta_t[b] = b_k$ et si $\nexists \tau \in \mathsf{AT}(t+1)$ tel que $\mathsf{ori}(\tau) = b_k \implies \zeta_{t+1}[b] = \zeta_t[b] = b_k$.

Dans la définition 3.15 ci-dessus, la première partie définit l'état d'un automate a après son changement par une transition locale temporisée $\tau = a_i \xrightarrow[\delta]{\ell} a_j$ activée à un instant t. Elle montre que ce changement n'est effectif (c'est-à-dire a change de a_i vers a_j) qu'à l'instant $t + \delta$. En effet, l'instant $t + \delta$ correspond à l'instant auquel τ se termine. Et la deuxième partie de la définition traite les états des automates non-changés : si un automate b n'a pas subit un changement par une des transitions locales temporisées, alors il garde dans l'état global successeur, ζ_{t+1}, son même niveau local actif dans ζ_t.

Dans cette section, nous avons parlé des délais intégrés dans les T-AN sous la forme d'entiers ($\delta \in \mathbb{N}$) et nous avons présenté la sémantique que nous adoptons pour l'étude de leurs dynamiques. Dans le chapitre 7, nous donnons une brève explication de la dynamique des T-AN si la composante temporelle est plutôt un rationnel ($\delta \in \mathbb{Q}$). En effet, ce travail est en cours et s'inscrit parmi les perspectives de cette thèse.

3.4 Les réseaux d'automates avec le temps par rapport à d'autres modèles temporels existants

L'enrichissement des formalismes de modélisation avec l'intégration d'une variable temporelle a fait l'objet de plusieurs résultats scientifiques que nous détaillons ci-dessous.

3.4.1 Frappes de processus

Dans (Paulevé et al., 2011*a*), les auteurs ont proposé un enrichissement des Frappes de Processus standards à l'aide de paramètres stochastiques. Les Frappes de Processus standards est une restriction des AN, présentée dans la section 2.5.1 en page 42 du chapitre 2. L'objectif était d'intégrer des données temporelles continues dans les modèles. Le modèle des T-AN intègre une composante temporelle discrète ($\delta \in \mathbb{N}$). De plus, leur enrichissement est directement inspiré du pi-calcul stochastique (Priami, 1995). Cependant, la loi exponentielle utilisée pour la simulation stochastique possède une trop grande variabilité, ainsi, l'approche a été raffinée par l'introduction d'un paramètre supplémentaire permettant de réduire l'intervalle de tir (Paulevé, Magnin & Roux, 2011*b*). Leur intervalle de tir, correspond à la durée de temps mise par la transition locale pour s'activer et que nous appelons en T-AN par le délai. Il est donc plutôt variable et non pas constant comme dans les T-AN.

3.4.2 Le paradigme $S[B]$

Des préoccupations similaires sont présentées dans le travail de (Merelli, Rucco, Sloot & Tesei, 2015), où les auteurs modélisent la dynamique du système en utilisant le paradigme $S[B]$ via un automate discret différé et une entropie persistante. Cependant, alors qu'ils se concentrent sur les périodes de temps écoulées entre les états globaux du système, nous visons à saisir les périodes de temps écoulées pour chaque interaction locale d'un automate (i.e., composant du système). Leurs méthodes peuvent reproduire l'évolution globale du système, mais ce que nous ciblons ici est l'évolution précise de chaque composant dans le système en chaque période de temps.

3.4.3 Les réseaux de Petri temporels et temporisés

On note que le formalisme des T-AN est beaucoup plus proche des réseaux de Petri (RdP) *temporisés* (Ramchandani, 1973) que des réseaux de Petri *temporels* (Merlin, 1974) . Dans les derniers, le temps est introduit par un délai minimal, autrement dit, ce délai représente le fonctionnement d'un "au plus tôt" du réseau permettant l'occurrence d'un événement. Par contre, dans les RdP temporisés, le temps est intégré sous forme d'un intervalle contraignant les instants de tir des transitions.

En effet, le temps se trouve aussi dans les transitions des RdP temporels, c'est ce qui est appelé *un intervalle de tir*. Par contre, dans les RdP temporisés, le temps est plutôt dans les états globaux du système. Les RdP temporels ont été initialement proposés comme purement asynchrone ; des transitions qui sont jouables dans le même état ne sont pas tirées en parallèle. Ceci n'est pas le cas dans la sémantique de la dynamique des T-AN que nous avons introduite dans ce chapitre. Bien sûr, à l'exception des transitions qui sont

en conflit (voir définition 3.6 en page 63) c'est-à-dire qui ne peuvent pas être activées en parallèle.

3.4.4 Chaîne de Markov(k)

D'autre part, la dynamique des T-AN peut être vue comme une sous-classe de la chaine de Markov(k). Nous donnons ci-après, une explications brève et informelle de cette considération.

Une chaîne de Markov est un processus de Markov à temps discret et à espace d'états discret. Un processus de Markov est un processus stochastique possédant la propriété de Markov : l'information utile pour la prédiction du futur est entièrement contenue dans l'état présent du processus et n'est pas dépendante des états antérieurs (système sans "mémoire"). En revanche, dans une chaine de Markov(k), l'information utile pour la prédiction du futur dépend des k états antérieurs (système avec "mémoire").

Considérant une chaine de Markov(k) représentée sous forme d'un ensemble de règles logiques, telle que chaque règle présente une prédiction de l'état d'un composant dans l'état suivant. Dans une règle (voir exemple ci-dessous), la valeur de l'atome dans la partie gauche (la tête de la règle) dépend des valeurs des atomes dans la partie droite (le corps de la règle). En effet, la partie droite représente la prédiction d'un état local d'un automate qui dépend des états antérieurs représentés dans la partie gauche de la règle.

Exemple. Quelques règles d'un programme logique représentant une chaine de Markov(k). La première règle dit que le niveau local de a sera égal à 1 au prochain état (à $t+1$), si b est au niveau 1 à l'état précédent $t-1$ et à l'état actuel t et si c est au niveau 1 à $t-3$ et à t. La deuxième règle dit que b sera au niveau 0 à $t+1$ si a est au niveau 1 à $t-2$ et aussi à $t-1$ et si b et c sont au niveau 1 à t.

```
a(1,t+1) ← b(1,t-1), b(1,t), c(1,t-3), c(0,t).
b(0,t+1) ← a(1,t-2), a(1,t-1), b(1,t), c(1,t).
```

Pour le formalisme des T-AN, chaque transition locale temporisée (τ) peut être représentée par une règle logique. En effet, la partie gauche de la règle représente $\mathsf{dest}(\tau)$ et la partie droite représente $\mathsf{cond}(\tau)$ et $\mathsf{ori}(\tau)$. Nous détaillons ci-dessous, un exemple d'une transition locale temporisée exprimée par une règle logique dans un système Markov(k).

Soit la transition locale temporisée suivante : $\tau = a_1 \xrightarrow[5]{\{b_1\}} a_0$.

Son équivalent en une chaine de Markov(k) tels que $t, t-1..., t-5$ sont des instants dans un chemin peut être représenté par la règle logique suivante :

```
a(0,t+1) ← a(1,t-4), b(1,t-4), a(1,t-3), b(1,t-3), a(1,t-2), b(1,t-2), a(1,t-1), b(1,t-1),
          a(1,t), b(1,t).
```

Cette règle logique exprime le fait que a passe du niveau local 0 vers 1 à l'instant $t+1$, quand $a=1$ et $b=1$ à tous les instants $t, t-1, ...$, jusqu'à $t-4$. Autrement dit, $a=1$ et $b=1$ pendant $\mathsf{delai}(\tau)$ unités de temps ; en l'occurrence ici 5 unités de temps. Ainsi, toute transition locale temporisée dans un T-AN peut être exprimée par une règle logique dans une chaine de Markov(k). Par conséquent, la dynamique d'un T-AN est une *sous-classe* de celle d'une chaine de Markov(k). En effet, ce dernier n'exige pas dans sa dynamique que les composants qui participent à l'activation d'une transition aient des niveaux constants tout au long de la période d'activation. Par exemple, b peut être au niveau 1 à $t-4$ et à $t-3$ mais il peut changer de niveau à $t-2$.

```
a(0,t+1) ← a(1,t-4), b(1,t-4), a(1,t-3), b(1,t-3), a(1,t-2), b(0,t-2), a(1,t-1), b(0,t-1),
          a(1,t), b(0,t).
```

Alors que dans la dynamique d'un T-AN, ce cas ne peut pas apparaître. Parce qu'on considère que si un composant est nécessaire pour l'activation d'une transition locale temporisée donnée, et s'il change de niveau au cours de l'activation de cette transition, cette dernière n'est plus jouable. Et ainsi, elle ne se termine pas et l'automate qui la contient ne change pas de niveau.

On note, de plus, une autre différence entre la sémantique de la dynamique des modèles T-AN et celle de la chaine de Markov(k) concernant la méthode du calcul d'un chemin. En effet, comme il a été montré dans la section précédente (par exemple dans le tableau 3.1 en page 76) pour simuler un T-AN et trouver son chemin, nous calculons à chaque instant t, l'ensemble des transitions locales temporisées franchissables, celles qui sont en cours et celles qui sont bloquées. Ainsi, à partir de ces ensembles, nous identifions l'ensemble des transitions à activer qui va définir les instants *"futurs"* auxquels les changements des niveaux locaux des automates se produisent. Par exemple, prenant la même transition locale temporisée ci-dessus, $\tau = a_1 \xrightarrow[5]{\{b_1\}} a_0$, si elle est activée à un instant t alors on sait d'avance que a changera de niveau à $t + 5$. Par contre, pour trouver une trajectoire d'une chaine de Markov(k), on doit regarder, à chaque instant t, les niveaux des automates dans le *"passé"*. En l'occurrence, pour cet exemple, on regarde les états de a et b à t, $t - 1$, ... jusqu'à $t - 4$ pour trouver l'état de a à $t + 1$.

3.5 Discussion

Notre contribution présentée dans ce chapitre, consiste à raffiner la dynamique des modèles discrets modélisés avec le formalisme des AN. Le raffinement est obtenu grâce à l'introduction des paramètres temporels : les *délais*. Un délai représente la période de temps nécessaire pour que des composants interagissent entre eux selon des niveaux d'expression précis, afin de provoquer le changement de l'un d'entre eux. Ce changement n'est qu'une activation ou inhibition du composant ciblé par cette interaction temporisée.

Nous appelons le formalisme ainsi enrichi "réseau d'automates avec le temps "(T-AN). Il est adapté à la modélisation des systèmes dynamiques complexes. Son utilisation pour la modélisation des RRB, n'écarte pas le fait qu'il peut être utilisé pour modéliser d'autres systèmes concurrents. D'ailleurs nous avons parlé dans la section 3.4, du positionnement des T-AN par rapport à d'autres formalismes existants, comme les réseaux de Petri temporisés et les chaines de Markov(k).

En outre, nous avons étudié la dynamique des T-AN qui concerne principalement la manipulation des transitions locales temporisées ayant des conflits entre elles. De plus, elle permet de raffiner la dynamique d'un AN (sans les délais) et ainsi de prévoir des comportements plus réalistes du modèle que celui présenté en AN.

Finalement, la dynamique permet aussi de concevoir des méthodes qui infèrent des T-AN à partir de leurs chemins obsevés. En fait, dans ce chapitre, nous avons le modèle, présenté avec le formalisme T-AN, et nous avons introduit la sémantique avec laquelle son évolution dynamique est trouvée (c'est-à-dire calculer son chemin). A contrario, dans le chapitre suivant, nous considérons que nous avons les chemins du modèle mais nous n'avons pas le modèle. Ainsi, le but est de réussir à proposer des méthodes qui infèrent le modèle T-AN à partir de ses chemins.

Chapitre 4

L'inférence des réseaux d'automates avec le temps

La modélisation des réseaux de régulation biologique (RRB) repose sur la connaissance de fond, dérivée de la littérature et/ou de l'analyse des données de séries temporelles issues des expériences biologiques. Nous comptons nous servir de ces dernières, dont la disponibilité n'est plus un problème grâce à l'évolution technologique, pour apporter une contribution à l'inférence du RRB qui modélise le système auquel les données appartiennent. Ainsi, Le but principal de ce chapitre est de présenter une nouvelle méthode d'inférence des RRB à partir des données de séries temporelles. Elle doit réussir à générer un RRB résultant aussi cohérent que possible avec la dynamique observée du système via ses données de séries temporelles. L'originalité de cette méthode repose principalement sur : (i) l'identification du signe des interactions entre les composants ; (ii) l'intégration directe des délais dans l'approche d'apprentissage (ces délais représentent la durée de temps nécessaire pour qu'une interaction se produise) ; et (iii) l'identification des niveaux discrets qualitatifs des composants participants à chaque interaction dans le réseau. Les RRB appris sont modélisés avec le formalisme des réseaux d'automates avec le temps (introduits dans le chapitre précédent). Nous étudions aussi les avantages d'une telle approche automatique par son application sur un ensemble des données synthétiques et sur des données réelles du système de l'horloge circadienne.

Ce travail a été publié dans (Ben Abdallah et al., 2016) puis dans une version étendue de ce travail dans (Ben Abdallah, Ribeiro, Magnin, Roux & Inoue, 2017).

4.1 Introduction

Durant la dernière décennie, l'émergence d'une large gamme de nouvelles technologies a produit une quantité massive de données biologiques (génomique, protéomique . . .). En effet,

avec le développement de nouvelles méthodes expérimentales, telles que les microarrays d'ADN en biologie, une très grande quantité de "*données de séries temporelles*" (i.e., des observations expérimentales) est maintenant produite.

Les données nouvellement produites peuvent nous fournir de nouvelles idées sur le comportement d'un système vivant. Cela conduit à des développements considérables dans la biologie des systèmes qui pourraient bénéficier de cette énorme quantité de données. Cela renforce la motivation pour développer des méthodes efficaces pour l'inférence des réseaux de régulation biologique (RRB).

Ainsi, nous présentons dans ce chapitre, une méthode qui vise à effectuer un apprentissage automatique des RRB. L'apprentissage est le processus qui traite les données de séries temporelles des composants du système étudié prises comme entrée afin de construire un RRB qui pourrait présenter d'une manière équivalente sa dynamique (voir figure 4.1 ci-dessous). Dans notre méthode, les RRB sont représentés avec le formalisme des réseaux d'automates avec le temps (T-AN). Ce dernier est un modèle discret introduit dans le chapitre précédent et qui s'est avéré adéquat pour la représentation des systèmes biologiques (Ben Abdallah et al., 2016; Ben Abdallah, Ribeiro, Magnin, Roux & Inoue, 2017). Ceci est aussi valable pour sa forme initiale sans l'intégration du temps, c'est-à-dire les réseaux d'automates (AN) (Paulevé et al., 2014).

C'est ce que nous développons dans la section 4.2 et nous appelons ceci l'inférence brute.

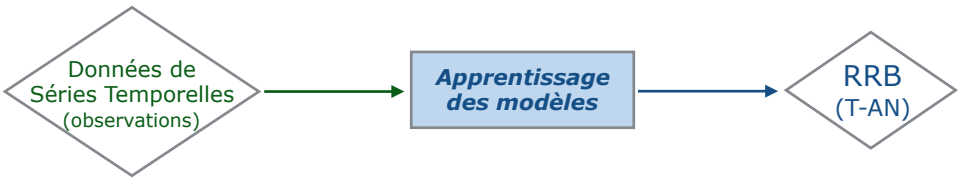


Figure 4.1 : La motivation de ce chapitre est de réussir à trouver une méthode automatique qui apprenne les RRB, représentés en T-AN, à partir des données de séries temporelles.

Puis dans la section 4.3, nous étudions le résultat de la méthode de l'apprentissage et nous validons la complétude de son algorithme.

Ensuite, dans la section 4.4, nous *raffinons* les modèles résultant de la nouvelle méthode d'apprentissage avec l'application d'un ensemble de "*filtres*". Ces filtres assurent un meilleur résultat vis-à-vis des modèles appris. Ils permettent d'éliminer les incohérences, s'il en existe, entre la dynamique d'un modèle appris par rapport à la sémantique de la dynamique des T-AN (introduite au chapitre 3). Un autre filtre permet de comparer tous les modèles appris entre eux et d'éliminer les modèles qui sont moins probablement corrects. Les critères qui permettent de calculer cette probabilité sont détaillés lors de la définition de ce filtre dans la section 4.4. Nous présentons aussi d'autres filtres qui contrecarrent à des informations incorrectes apprises dans les modèles et qui sont dues aux données bruitées.

Nous montrons à la fin de cette section l'utilité de ce raffinement des modèles appris par l'algorithme d'apprentissage et nous comparons les résultats obtenus sans et avec les filtres. Pour cela, nous appliquons ces méthodes sur des données synthétiques issues des modèles biologiques réels.

Puis, nous discutons dans la section 4.5, le fait que l'algorithme d'apprentissage proposé dans ce chapitre peut être adapté pour qu'il soit capable de réviser des modèles existants.

En effet, nous considérons alors qu'un modèle préliminaire existe déjà et que des données de séries temporelles issues de nouvelles expériences sont fournies. Ainsi, le but est de réviser ce modèle pré-existant pour assurer que sa dynamique est cohérente avec celle exprimée dans les données de séries temporelles nouvellement fournies.

Finalement, dans la section 4.6, nous appliquons la méthode d'apprentissage sur des données réelles issues des mesures sur le foie des souris pour étudier le système de l'horloge circadienne. Ensuite, nous discutons dans la section 4.7, les méthodes et les résultats que nous avons présentés dans ce chapitre, et nous nous positionnons par rapport à quelques autres méthodes existantes.

4.2 La méthode d'inférence brute

Dans cette section, nous proposons une méthode d'apprentissage automatique qui construit un T-AN modélisant un RRB à partir de ses données de séries temporelles. Ces données expriment l'évolution temporelle de l'expression des composants du système étudié. Autrement dit, elles permettent de visualiser au cours du temps, à chaque instant, le niveau d'expression de chaque composant du système. Il s'agit d'une méthode brute car elle sera ultérieurement raffinée par des filtres en section 4.4.

Nous expliquons dans la sous-section 4.2.1 suivante, l'intérêt de pré-traiter les données de séries temporelles avant de les fournir à l'algorithme d'apprentissage. En effet, ce pré-traitement permet principalement d'avoir des données qui sont manipulables par l'algorithme.

Ensuite, nous introduisons dans la sous-section 4.2.2 en page 91, l'algorithme d'apprentissage. De plus, nous vérifions sa propriété de complétude par rapport aux modèles appris. Enfin, nous validons le résultat de l'algorithme par la comparaison des modèles appris à partir des données de séries temporelles synthétiques issues des T-AN modélisant des systèmes biologiques réels.

4.2.1 Pré-traitement des données de séries temporelles

D'une façon générale, les données expérimentales sont des données générées dans le cadre d'une étude scientifique par observation et enregistrement. L'objet des données de séries temporelles est l'étude de l'évolution des variables au cours du temps. Il est à noter qu'actuellement, les données ont été décrites comme le nouveau pétrole de l'économie numérique.

Dans le cadre de la biologie des systèmes, les données de séries temporelles représentent des mesures de l'expression des composants biologiques (gènes, ADN, ARNm, protéines...). Elles correspondent à des valeurs des concentrations biologiques obtenues par des mesures faites par des outils dédiés selon une période de temps régulière ou non. De nos jours, on remarque une généralisation des outils numériques consacrés aux mesures des composants biologiques, comme c'est indiqué dans (Git, Dvinge, Salmon-Divon, Osborne, Kutter, Hadfield, Bertone & Caldas, 2010). Par exemple, on note le développement des méthodes NGS (Next Generation Sequencing) telles que les données des microarrays d'ADN (Marx, 2013).

Dans la figure 4.2 en page 88, les courbes en noir montrent un exemple de données de séries temporelles pour deux composants x et y. Principalement dans ce chapitre, les données de séries temporelles que nous utilisons dans la suite de nos travaux sont de ce

type. Nous détaillons dans la suite les différents traitements que nous effectuons sur ces données.

4.2.1.1 La discrétisation des données de séries temporelles

En général, une *discrétisation* est l'opération qui permet de découper en classes une série de variables qualitatives ou de variables quantitatives. Sur les données de séries temporelles, cette opération simplifie l'information en regroupant les concentrations d'un composant biologique présentant les mêmes caractéristiques en classes distinctes. Nous appelons alors ces classes par des *niveaux de discrétisation*. On considère, qu'une discrétisation est satisfaisante lorsqu'elle permet la création de classes homogènes et distinctes entre elles.

Discrétiser des données de séries temporelles constitue souvent l'ultime étape de la réduction, de l'organisation et de la hiérarchisation de l'information avant de construire un modèle discret qui représente le système auquel les données de séries temporelles appartiennent.

Dans l'idéal, l'opération de la discrétisation doit préserver les propriétés emblématiques du système biologique étudié. Elle doit conserver les caractéristiques essentielles présentées par les données, et perdre le moins d'information possible, afin de transmettre une information efficace et de qualité sur la dynamique biologique du système.

Il existe un grand nombre d'approches de discrétisation. Le choix d'une approche dépend à la fois des propriétés du système étudié et des objectifs que l'on s'est fixés quant à l'information à communiquer. Une méthode peut être choisie selon comment elle traite les données par rapport à un critère spécifique. Par exemple, la caractérisation de la vitesse de la variation de l'expression des concentrations des composants (rapide, lente, très rapide ou très lente), ou du sens de la variation de la concentration (augmentation, diminution ou stagnation). Une autre méthode de discrétisation consiste à définir pour chaque classe des valeurs des bornes limites (i.e., une valeur minimale et une valeur maximale). Ainsi, dans tous les cas, quel que soit le critère choisi, il y a la définition d'un *seuil* entre les classes tel que ce seuil fait référence à des critères explicitement définis. Et donc, afin d'avoir une bonne discrétisation, il faut justifier le choix de ces seuils.

Par conséquent, rendre discret les données de séries temporelles revient à dégager des valeurs qualitatives à partir des données continues quantitatives. Ces valeurs qualitatives, qui sont propres à chaque composant du système étudié, correspondent à ses niveaux discrets dans le modèle représentant le système (voir définition 4.1 ci-dessous inspirée de (DeCarlo, 1989)).

Définition 4.1 (Discrétisation)**.** La *discrétisation* est le processus de transformation des fonctions continues en leurs homologues discrètes.

Dans notre cas, les fonctions continues sont les données de séries temporelles des composants du système étudié. Ainsi, la discrétisation consiste à transformer les données de séries temporelles exprimées avec des valeurs *réelles* en des données exprimées avec des valeurs *entières*. On a alors comme résultat, une *fonction en escalier* qui représente l'évolution discrète de chaque composant (voir les courbes en bleu dans la figure 4.2 en page 88). En effet, une fois définis les seuils, qui délimitent un niveau discret, la projection sur l'axe du temps de la courbe des données de séries temporelles définit un intervalle de temps auquel on associe ce niveau discret qualitatif.

Nous appelons ces fonctions en escalier des *chronogrammes* (voir définition 4.2 ci-dessous).

Définition 4.2 (Chronogramme)**.** Un *chronogramme* est le résultat de la discrétisation des données de séries temporelles continues de chaque composant du système étudié. Il est défini par la fonction Γ ci-dessous :

$$\Gamma : [0, T] \subset \mathbb{N}^+ \longrightarrow \{0, ..., n\}$$
$$[t_1, t_2[\quad \longmapsto \quad i$$

tel que, t_1, $t_2 \in \mathbb{N}$, $t_1 < t_2 \leq T$ et $\forall t \in [t_1, t_2[$:

$$\Gamma(t) = \Gamma(t_1) = i \quad \text{et}$$

$$\Gamma(t_2) = i + 1 \text{ ou } \Gamma(t_2) = i - 1$$

où T est le dernier point temporel dans les données de séries temporelles ; ce qui correspond à la valeur maximale sur l'axe du temps. T est appelé la *taille* du chronogramme et n est le niveau discret maximal.

On note par Γ_a le chronogramme trouvé à partir des données de séries temporelles du composant a. Nous donnons ci-dessous un exemple qui montre la discrétisation de deux courbes correspondant à des données de séries temporelles des composants x et y selon des seuils pré-définis (choisis selon certains critères).

Exemple. Les courbes en noir de la figure ci-dessous, représentent des exemples des données de séries temporelles des composants x et y. Et les courbes en bleu représentent les résultats de leurs discrétisations ; Γ_x et Γ_y les chronogrammes respectifs de x et y. On note que pour cet exemple, le choix des seuils pour la discrétisation est positionné d'une manière arbitraire.

En effet, comme le montre la figure, x a un seul seuil, noté "seuil". Il sépare les données de séries temporelles de x en 2 niveaux discrets distincts : 0 et 1. Par contre, le composant y a deux seuils "seuil 1" et "seuil 2". Ce qui justifie le fait qu'il a trois niveaux discrets dans le chronogramme Γ_y : 0, 1 et 2.

On remarque que la taille de ces deux chronogrammes Γ_x et Γ_y est égale à $T = 20$. Donc l'intervalle de la définition des deux chronogrammes est $[0, T] = [0, 20]$. Ainsi, comme c'est indiqué dans la définition 4.2 ci-dessus, on associe à chaque sous-intervalle $[t_1, t_2[\subseteq [0, 20]$, un niveau discret. À titre d'exemple, dans Γ_x, on trouve que :

$$[0, 12[\xmapsto{\Gamma_x} 1 \quad \text{et} \quad [12, 20] \xmapsto{\Gamma_x} 0$$

et dans Γ_y :

$$[0, 2[\xmapsto{\Gamma_y} 0, \quad [2, 13[\xmapsto{\Gamma_y} 1 \quad \text{et} \quad [13, 20] \xmapsto{\Gamma_y} 2.$$

Pour plus de précision, on associe à chaque instant $t \in [0, T]$ la même valeur associée à l'intervalle où il est inclus. C'est-à-dire, si $t \in [T_1, T_2[$ et si $[T_1, T_2[\xmapsto{\Gamma} i \in \mathbb{N}$ pour le chronogramme Γ, alors $t \xmapsto{\Gamma} i$. En l'occurrence, pour les chronogrammes Γ_x et Γ_y, si $t = 17$, on a $\Gamma_x(17) = 0$ car $17 \in [12, 20[$ et $\Gamma_y(17) = 2$ car $17 \in [13, 18[$.

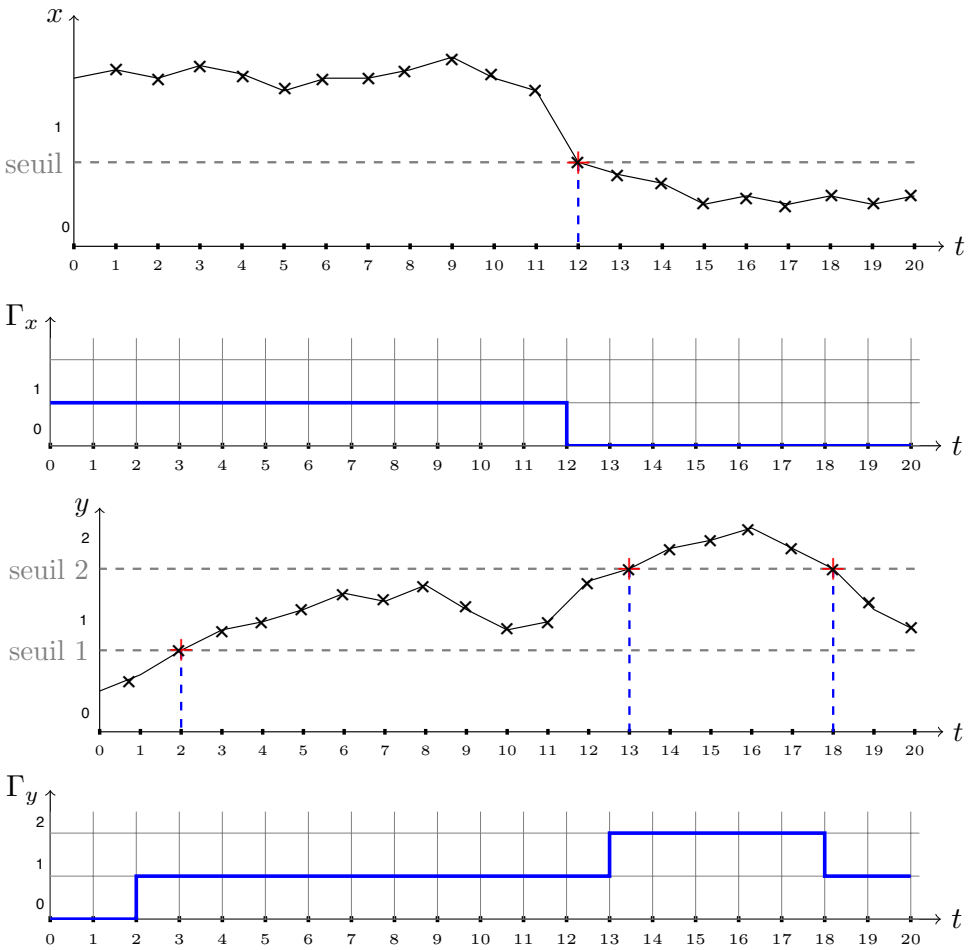


Figure 4.2 : Exemples de données de séries temporelles des composants x et y avant (courbe en noir) et après (chronogramme en bleu) la discrétisation.

Ayant ces chronogrammes comme entrée, l'algorithme d'apprentissage des modèles —que nous détaillons dans la section 4.2.2 suivante— infère des informations nécessaires pour trouver le modèle T-AN qui modélise le système étudié. Ces informations inférées sont d'une part sur la durée de temps nécessaire pour que chaque transition responsable du changement d'un composant d'un niveau discret vers un autre ait lieu. D'autre part, elles concernent les niveaux discrets des composants impliqués dans cette transition.

Nous notons que l'algorithme prend aussi une autre information en entrée. En effet, il est mieux de connaître au préalable les différentes interactions qui existent entre les composants du système. Autrement dit, il faut identifier au préalable quels sont les composants qui sont responsables de la régulation de chaque composant du système. Cette information est capturée par ce que l'on appelle le *graphe d'influences*. En fait, ce graphe facilite le processus de la recherche pour trouver quels sont les composants qui sont responsables de chaque changement dans la dynamique du système. En plus, il permet de diminuer sa complexité tout en étant plus précis par l'identification des transitions locales temporisées du T-AN appris.

4.2.1.2 Le graphe d'influences

Un second pré-traitement des données de séries temporelles, consiste à obtenir le graphe d'influences entre les composants du système étudié. En effet, une fois que ce graphe est trouvé, notre but est alors de le raffiner à partir des données de séries temporelles discrétisées, par l'identification (i) des niveaux discrets des composants participant à chaque interaction, (ii) du signe de l'interaction et (iii) de son délai (la durée nécessaire pour que son activité ait un effet).

Dans la suite, nous donnons la définition formelle d'un graphe d'influences suivi par un exemple.

Définition 4.3 (Graphe d'influences)**.** Un graphe d'influences est un graphe orienté $\chi(N, E)$ tel que :

- N est l'ensemble de tous les nœuds représentant les composants du système et
- $E \subseteq N \times N$ est l'ensemble des arcs du graphe tels que si $(x, y) \in E$ avec x, $y \in N$ alors x exerce une *influence* sur y. Cette influence peut être activatrice ou inhibitrice.

On note $\chi_a(N_a, E_a)$ le "sous graphe" d'influences de "a", autrement dit, la partie du graphe d'influences dont la destination de tous ses arcs est le composant "a". Ainsi, χ_a ne contient que les régulateurs de a (voir propriété 4.1 ci-dessous).

Propriété 4.1. *Soit $\chi_a(N_a, E_a)$ le sous-graphe d'influences de a tel que $N_a \subseteq N$ et $E_a \subseteq E$ avec $E_a \subseteq N_a \times a$. Si $x \in N_a$ alors $\exists e \in E$ tel que $e = (x, a)$.*

Ainsi, on appelle N_a et E_a respectivement l'ensemble des régulateurs *de a et l'ensemble des* actions régulatrices *sur a.*

Exemple. Dans la figure 4.3, nous donnons un exemple de graphe d'influences $\chi(N, E)$ tel que : $N = \{a, b, z\}$ et $E = \{(a, a), (z, a), (a, b), (a, z), (b, z)\}$.

Nous en déduisons alors les ensembles χ_a, χ_b et χ_z définis ci-dessous :

- $\chi_a(N_a, E_a)$ avec $N_a = \{a, z\}$ et $E_a = \{(a, a), (z, a)\}$,
- $\chi_b(N_b, E_b)$ avec $N_b = \{a\}$ et $E_b = \{(a, b)\}$,
- $\chi_z(N_z, E_z)$ avec $N_z = \{a, b\}$ et $E_z = \{(a, z), (b, z)\}$.

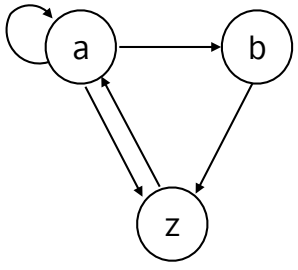


Figure 4.3 : Un graphe d'influences χ dont les nœuds sont : "a", "b" et "z" et représentent les composants du système modélisé. Chaque arc orienté représente une action régulatrice du composant régulateur (origine de l'arc) sur le composant régulé (destination de l'arc).

Dans certains cas, l'influence d'un unique composant x sur un autre y ($(x, y) \in E$) seule n'est pas suffisante pour qu'il y ait un effet régulateur sur y. En effet, elle nécessite

la présence (ou l'absence) d'une ou de plusieurs autres influences en même temps. Par exemple, pour activer y, il faut que x et z soient tous les deux activés au même instant tel que $\exists(z, y) \in E$. De plus, il faut que w soit inactif. Ainsi, la situation dans laquelle l'interaction sur y se produit, peut être décrite par la formule propositionnelle logique suivante : $m = x \wedge z \wedge \neg w$ où les symboles $\wedge$ et $\neg$ expriment respectivement un "et logique" et un "non logique". Cette formule montre la coopération entre x, z et $\neg w$ pour activer y. m est appelé un "multiplexe" dans le mécanisme présenté dans (Khalis, Comet, Richard & Bernot, 2009).

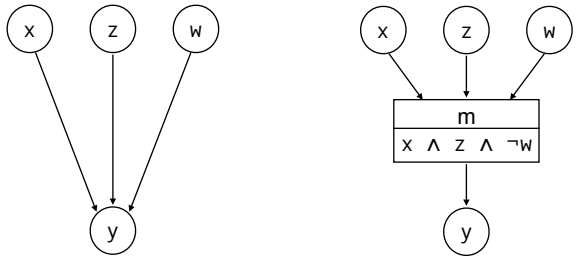


Figure 4.4 : Exemple d'une influence avec coopération entre 3 nœuds x, z et w illustrée par le multiplexe libellé m tel que $m = x \wedge z \wedge \neg w$.

Puisque chaque composant pourrait avoir plusieurs régulateurs, il est nécessaire d'identifier les multiplexes corrects qui agissent sur lui et qui causent son changement d'un niveau discret vers un autre. Dans le formalisme des T-AN, pour chaque transition locale temporisée τ, le multiplexe est représenté par la condition de cette transition, $\mathsf{cond}(\tau)$ (voir section 3.2 du chapitre précédent en page 55). Par exemple, si on suppose que le niveau discret 1 représente l'état local actif d'un composant et 0 son état local inactif, on peut présenter le multiplexe de la figure 4.4 ci-dessus par la transition locale temporisée suivante : $\tau = y_0 \xrightarrow[\delta]{\{x_1, z_1, w_0\}} y_1$ où $\delta \in \mathbb{N}$. En effet, pour activer y, x et z doivent être tous les deux présents, alors que w doit être absent. Ainsi, $\mathsf{cond}(\tau) = \{x_1, z_1, w_0\}$ représente dans un T-AN le multiplexe $m = x \wedge z \wedge \neg w$.

Nous avons trouvé que dans la littérature, il existe plusieurs moyens d'identification un graphe d'influences pour un système donné. Les méthodes les plus répandues sont celles qui calculent la corrélation entre les composants à partir des données de séries temporelles. Par exemple, dans (Villaverde, Becker & Banga, 2016) les auteurs présentent un outil appelé PREMER qui permet de récupérer le graphe d'influences du système en estimant la causalité des interactions entre les composants du système et en se basant sur des critères théoriques de l'information.

D'autre part, de nos jours plusieurs bases de données, où de nombreux modèles des systèmes biologiques sont sauvegardés, sont publiques et accessibles en ligne. À partir de ces bases de données, il est possible d'extraire des informations à propos des régulations entre des composants biologiques d'un système biologique quelconque.

Ces bases de données englobent aussi des modèles des réseaux construits par des biologistes et/ou des bioinformaticiens. Nous citons à titre d'exemples des bases de données spécialisées dans les connaissances de réseaux des régulations chez l'humain : Human

Integrated Pathway Database (Yu, Seo, Rho, Jang, Park, Kim & Lee, 2012) et Causal Biological Network Database (Talikka, Boue & Schlage, 2015).

La figure 4.5 ci-dessous, raffine la figure 4.1 en page 84 qui illustre la procédure présentée dans ce chapitre. En effet, nous avons subdivisé la phase d'apprentissage des modèles en deux. Une première phase consiste au pré-traitement des données dont le but est de discrétiser les données de séries temporelles et de trouver le graphe d'influences (comme il est montré dans cette sous-section). La deuxième phase consiste à prendre ces données pré-traitées en entrée à une méthode d'apprentissage des modèles qui génère des RRB (présentés avec le formalisme des T-AN). Nous introduisons dans la sous-section suivante l'algorithme de cette méthode.

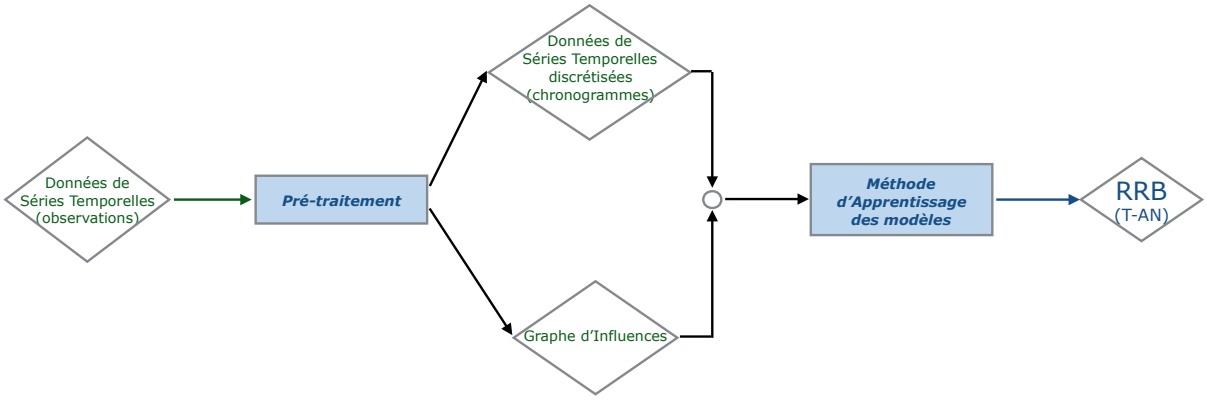


Figure 4.5 : Le pré-traitement des données de séries temporelles avant les fournir à la méthode d'apprentissage des modèles.

4.2.2 Algorithme de la méthode d'apprentissage des modèles : `MoT-AN`

Dans cette section, nous proposons un algorithme pour construire un T-AN à partir des données de séries temporelles. Nous appelons cet algorithme, par `MoT-AN` (Modélisation des T-AN).

Tout d'abord, nous introduisons les entrées de l'algorithme ; il prend comme première entrée des données de séries temporelles discrétisées, c'est-à-dire des chronogrammes (voir définition 4.2 en page 87), et la deuxième entrée est le graphe d'influences (définition 4.3 en page 89). Comme il est indiqué précédemment, le graphe d'influences peut être trouvé par la fouille des bases de données ou par des méthodes automatiques qui calculent la corrélation entre les composants du système à modéliser à partir de ses données de séries temporelles (Villaverde et al., 2016).

L'algorithme prend aussi en entrée un modèle présenté en T-AN (définition 3.7 dans le chapitre 3 en page 65), dont l'ensemble des transitions locales temporisées est *vide* ; le T-AN ne contient alors que des automates (représentant les composants du système) et leurs états locaux (déduits à partir de leurs niveaux discrets dans les chronogrammes).

Ainsi, l'objectif de `MoT-AN` est de remplir le vide dans l'ensemble des transitions locales temporisées du T-AN pris en entrée. Compte tenu des influences entre les composants (ou en supposant toutes les influences possibles si aucune connaissance à priori n'est disponible), `MoT-AN` génère les transitions locales temporisées qui pourraient entraîner la même dynamique que celle observée dans les données de séries temporelles prises sous la forme des chronogrammes.

En effet, MoT-AN trouve toutes les transitions locales temporisées candidates telles qu'elles peuvent produire chaque *changement* dans la dynamique. Un changement dans la dynamique (observée à travers les chronogrammes) correspond à une modification du niveau discret d'un composant du système (un automate en T-AN). Il se produit à cause d'une régulation activatrice ou inhibitrice sur le composant modifié qui est présenté en T-AN par une transition locale temporisée. Par conséquent, MoT-AN trouve ces transitions locales temporisées dont chacune s'écrit sous la forme suivante : $\tau = a_i \xrightarrow[\delta]{\ell} a_j$, en identifiant son origine ($\mathsf{ori}(\tau) = a_i$), sa destination ($\mathsf{dest}(\tau) = a_j$), tous les automates inclus dans sa condition ($\mathsf{cond}(\tau) = \ell$) et finalement son délai ($\mathsf{delai}(\tau) = \delta$) qui dépend de chaque automate impliqué dans τ.

Par la suite, trouver toutes ces informations permet d'en déduire les signes des régulations responsables de chaque changement dans la dynamique du système. En effet, si dans $\tau = a_i \xrightarrow[\delta]{\ell} a_j$, $i < j$ alors il s'agit d'une régulation activatrice sinon si $i > j$ c'est une régulation inhibitrice.

Nous donnons dans la suite (en page 93), le pseudo-code de l'algorithme MoT-AN. Plus loin dans ce manuscrit, au chapitre 6, nous donnons son implémentation en Answer Set Programming (ASP).

Pour une meilleure compréhension du pseudo-code, nous introduisons dans la remarque suivante quelques notations.

Remarque. Pour calculer le délai d'une transition locale temporisée qui a produit un changement de niveau d'un composant a à un instant t, il faut toujours se projeter dans le passé de la dynamique du système (c'est-à-dire avant d'atteindre l'instant t) et voir ce qui s'est passé pour causer ce changement. En général, ce sont les changements de niveaux des régulateurs de a qui provoquent le changement de niveaux de a. Ainsi, nous avons besoin d'identifier tous les changements de niveaux des régulateurs de a qui se sont produits avant celui de a qui s'est produit à l'instant t. On note alors par Π_t^x le *dernier* instant ayant produit un changement du composant x avant d'arriver à l'instant t ; i.e., $\forall x \in \Sigma$, $\Pi_t^x < t$. On suppose qu'à $t = 0$, tous les composants du système à modéliser commencent dans un état comme s'ils venaient d'être changés.

Exemple. Prenons l'exemple des chronogrammes des composants x et y sur les 20 unités de temps dans la figure 4.2 en page 88. Pour $t = 5$, on a $\Pi_5^x = 0$ et $\Pi_5^y = 2$ sont respectivement les derniers changements de x et y avant $t = 5$.

Et pour $t = 17$ on a : $\Pi_{17}^x = 12$ et $\Pi_{17}^y = 13$.

Dans la suite, la notation Π_t^x — l'instant du dernier changement du composant x avant d'arriver à l'instant t — est utile pour faciliter la compréhension du pseudo-code de l'algorithme 1, MoT-AN (ci-dessous).

Algorithm 1 `MoT-AN` : Modélisation des réseaux d'automates avec le temps.

Entrées :

- $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$: un T-AN avec $\mathcal{T} = \emptyset$;
- $\Gamma = \bigcup_{a \in \Sigma} \Gamma_a$: chronogrammes des données de séries temporelles ;
- $\chi(N, E) = \bigcup_{a \in \Sigma} \chi_a(N_a, E_a)$: graphe d'influences entre les automates de $\mathcal{AN}$;
- $n \in \mathbb{N}^*$: le nombre maximal des régulateurs sur un composant dans une transition locale temporisée apprise.

Sortie : Φ : l'ensemble des T-AN qui réalisent les chronogrammes Γ.

– Soit $\mathcal{T}_{appris} := \emptyset$ *//ensemble de toutes les transitions apprises*

– Étape 1 :

pour chaque point temporel t dans Γ où un composant a **change** son niveau discret de i vers j **faire** *//en T-AN, c'est le changement de l'état local a_i vers a_j*

- **pour chaque** $\ell \in \wp(N_a), |\ell| \leq n$ **faire**
  //pour chaque combinaison possible des régulateurs de a

$$\tau := a_i \xrightarrow[\delta]{\ell} a_j$$

telle que,

si $\exists\ \tau'{=}c_k \xrightarrow[\delta_c]{\ell_c} c_l \in \varphi$ avec $a_i \in \ell_c$, $\Pi_t^c \geq \Pi_t^a$, $\forall b_x \in \ell$, $\Pi_t^c \geq \Pi_t^b$ **alors**

//τ est bloquée par τ' donc τ n'est activée que quand τ' se termine à Π_t^c

$$\delta = t - \Pi_t^c$$

sinon *//τ est activée à partir de l'instant qui est égal au maximum des //instants entre les derniers changements de a et des automates de ℓ*

$$\delta = t - max(\Pi_t^a, \Pi_t^d)$$

avec $d_x \in \ell \wedge \forall b_x \in \ell,\ b \neq d,\ \Pi_t^b \leq \Pi_t^d$.

ajouter τ dans $\mathcal{T}_{change(t)}$.

//$\mathcal{T}_{change(t)}$ est l'ensemble des transitions apprises par changement à l'instant t

- **ajouter** toutes les transitions locales temporisées de $\mathcal{T}_{change(t)}$ dans $\mathcal{T}_{appris}$.

– Étape 2 :

Créer autant que possible dans Φ des T-AN appris, $\mathcal{AN}^k_{appris} = (\Sigma, \mathcal{S}, \mathcal{T}^k_{appris})$ avec $k \leq |\Phi|$ et $|\Phi|$ est le nombre total des T-AN appris. $\mathcal{T}^k_{appris} \subseteq \mathcal{T}_{appris}$ est l'ensemble **minimal** des transitions locales temporisées qui pourraient réaliser Γ, c'est-à-dire :

$$\forall \mathcal{AN}^k_{appris} = (\Sigma, \mathcal{S}, \mathcal{T}^k_{appris}) \in \Phi, \nexists \mathcal{T}^{k'}_{appris} \subset \mathcal{T}^k_{appris} \text{ tel que } \mathcal{T}^{k'}_{appris} \text{ peut réaliser } \Gamma,$$

autrement dit,

$\forall t$ un point temporel dans Γ, si $\exists a \in \Sigma$ tel que a change de niveau à t alors,

$$|\mathcal{T}^k_{appris} \cap \mathcal{T}_{change(t)}| = 1.$$

Par conséquent, l'algorithme 1, MoT-AN, génère un ensemble de T-AN qui satisfont la dynamique exprimée par les données de séries temporelles sous la forme des chronogrammes. Nous donnons dans la section 4.3 suivante, un exemple d'application de cet algorithme pour mieux comprendre ses étapes tout en proposant une méthode qui valide son résultat.

4.3 Validation du résultat de MoT-AN

Nous montrons dans cette section, la preuve pratique que l'algorithme 1, MoT-AN, est capable de retrouver toutes les transitions locales temporisées qui sont effectivement responsables des changements dans la dynamique du système à modéliser.

4.3.1 Démarche de validation pratique du résultat de MoT-AN

La figure 4.6 ci-dessous, résume les étapes de l'approche de validation du résultat retourné par la méthode d'apprentissage des modèles, MoT-AN. Nous partons d'un T-AN, que nous connaissons au préalable appelé "T-AN initial" (voir figure 4.6 ci-dessous). Le T-AN initial modélise un système biologique quelconque.

Afin d'avoir des chronogrammes illustrant l'évolution dynamique d'un système, nous simulons le T-AN initial qui le modélise en respectant la sémantique de la dynamique des T-AN (telle qu'elle est introduite dans la section 3.3.1 du chapitre 3 en page 65). Nous notons que pour chaque simulation, l'état global initial du T-AN est choisi aléatoirement. Nous rappelons que la méthode d'apprentissage, MoT-AN, prend les données de séries temporelles sous forme de chronogrammes, ainsi, nous avons développé une méthode qui simule les T-AN (en occurrence le T-AN initial) et retourne leurs évolutions dynamiques sous forme de chronogrammes. La taille des chronogrammes correspond au nombre maximal des instants atteints après l'état initial. Ce nombre est aussi choisi aléatoirement pour chaque simulation.

La deuxième entrée de MoT-AN est le graphe d'influences. Ce dernier est facilement trouvé à partir du T-AN initial en parcourant toutes ses transitions locales temporisées (noté $\mathcal{T}_{initial}$) : $\forall \tau \in \mathcal{T}_{initial}$ telle que $\tau = a_i \xrightarrow[\delta]{\ell} a_j$ et $\forall b_k \in \ell$, $\exists (b, a) \in E$ avec $\chi(N, E)$ est le graphe d'influences du T-AN initial.

Ainsi, en trouvant toutes les entrées nécessaires pour exécuter MoT-AN, il est possible de l'exécuter pour commencer l'apprentissage à partir des chronogrammes et du graphe d'influences du T-AN initial. Le but est alors de vérifier si MoT-AN réussit à trouver le T-AN initial à partir de ses chronogrammes. Autrement dit, réussir à apprendre le T-AN qu'il faut qu'il apprenne.

Une fois que nous récupérons tous les T-AN appris, nous prenons l'union de leurs ensembles de transitions locales temporisées (noté $\mathcal{T}_{appris}$) puis nous comparons l'ensemble trouvé de l'union à l'ensemble des transitions locales temporisées du T-AN initial (noté $\mathcal{T}_{initial}$). La comparaison consiste à vérifier si la totalité des transitions locales temporisées du T-AN initial ont effectivement été apprises par MoT-AN. Autrement dit, nous vérifions si $\mathcal{T}_{initial} \subseteq \mathcal{T}_{appris}$.

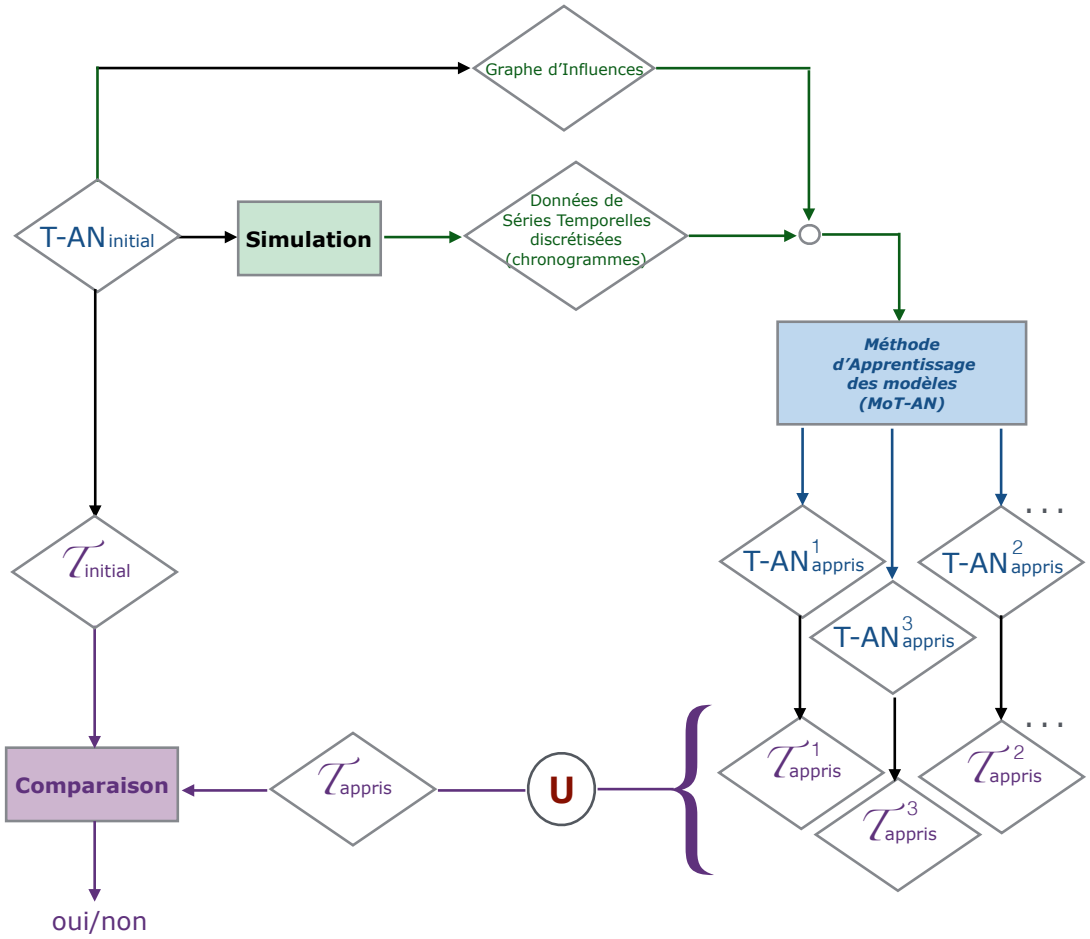


Figure 4.6 : Les étapes de la démarche de la validation de la méthode d'apprentissage des modèles T-AN : `MoT-AN`.

Nous montrons dans la suite une application de cette démarche de validation pratique du résultat de `MoT-AN` sur un petit exemple jouet d'un T-AN initial. Cet exemple permet aussi de montrer comment l'apprentissage avec `MoT-AN` est réalisé.

4.3.2 Exemple d'application de la démarche de validation pratique du résultat de `MoT-AN`

Pour montrer comment la démarche de validation est faite, nous reprenons dans la figure 4.7 ci-dessous, le modèle T-AN de l'exemple jouet de la figure 3.5 en page 66 du chapitre 3. Nous considérons cet exemple de T-AN étant le T-AN initial. Nous le simulons alors sur 18 unités de temps selon la sémantique des T-AN introduite dans le chapitre 3, section 3.3.1 en page 65. Une des simulations trouvée est présentée dans la figure 4.8 ci-dessous sous forme de chronogrammes. Chaque chronogramme présente l'évolution du niveau discret d'un composant du réseau. Nous notons que ces chronogrammes correspondent au chemin présenté dans le tableau 3.1 de la section 3.3.1 en page 76 du chapitre 3 où nous donnons aussi les détails de la démarche du calcul de ce chemin.

Dans la suite, nous exécutons l'algorithme 1, `Mot-AN`, étape par étape dont le pseudo-code est donné en page 93. Nous prenons comme entrée les chronogrammes de la figure 4.8 et le graphe d'influences de la figure 4.3 en page 89. Ensuite, nous vérifions si les transitions locales temporisées du modèle initial — à partir duquel les chronogrammes sont

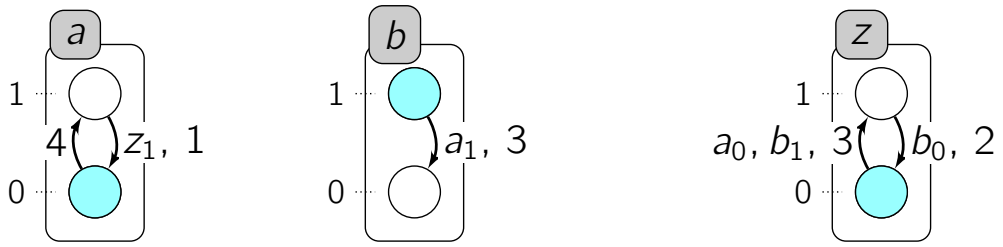


Figure 4.7 : Rappel de l'exemple jouet : un réseau d'automates avec 3 automates ayant chacun 2 états locaux. Ce modèle contient 5 transitions locales temporisées.

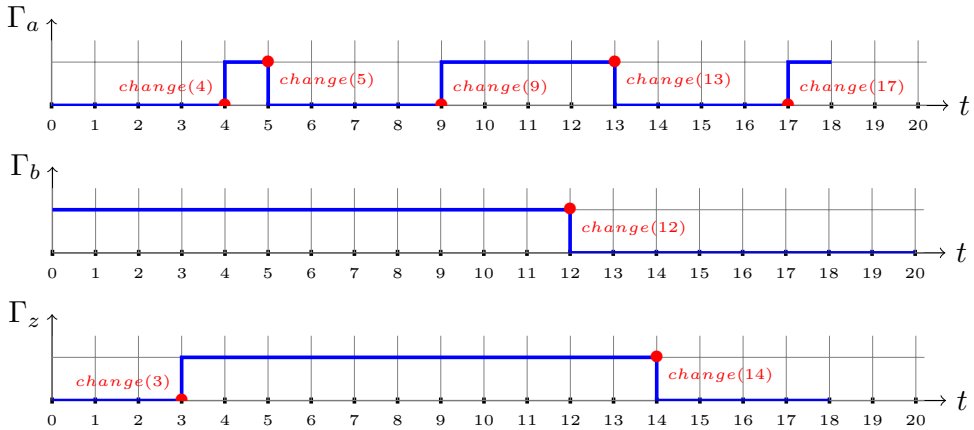


Figure 4.8 : C'est le résultat de l'étape simulation de la figure 4.6 en page 95 : en effet, ces chronogrammes sont trouvés par la simulation du T-AN initial (qui est l'exemple jouet de la figure 4.7) sur 18 unités de temps. Les chronogrammes Γ_a, Γ_b et Γ_c correspondent respectivement aux trois composants a, b et c. Chaque point rouge indique un changement du niveau discret d'un composant à un instant donné. Par exemple, à l'instant $t = 4$, *change(4)* indique le changement de a du niveau 0 au niveau 1.

calculés et en l'occurrence ici le T-AN de la figure 4.7 — existent aussi dans l'ensemble des transitions apprises. Puis vérifier aussi si l'un des T-AN appris est égal au T-AN initial. En effet, si c'est le cas, on peut prouver la validation pratique de la complétude de la méthode MoT-AN.

Comme le montrent les chronogrammes de la figure 4.8, le premier changement se produit à $t_{min} = t_1 = 3$, noté par $change(3)$. C'est le gène z dont le niveau discret passe de 0 à 1. En T-AN, ceci se traduit par le changement de l'état local z_0 vers z_1. Ainsi, la transition locale temporisée responsable de ce changement est de la forme : $\tau = z_0 \xrightarrow[\delta]{\ell} z_1$, où ℓ peut être n'importe quelle combinaison des régulateurs de z tels que ces régulateurs sont trouvés à partir du graphe d'influences associé au réseau.

On rappelle les éléments du graphe d'influences de cet exemple $\chi(N, E)$ pris de la figure 4.3 en page 89 : $N = \{a, b, z\}$ et $E = \{(a, a), (z, a), (a, b), (a, z), (b, z)\}$. Ainsi, l'ensemble des régulateurs de z noté N_z est égal à $N_z = \{a, b\}$.

Ce qui implique que pour ce premier changement à $t_1 = 3$, et pour la transition candidate d'en être responsable $\tau = z_0 \xrightarrow[\delta]{\ell} z_1$, on a $\ell = \mathsf{cond}(\tau)$ qui pourrait être égal à : $\ell = \{a_?, b_?\}$, ou $\ell = \{a_?\}$, ou $\ell = \{b_?\}$. Les points d'interrogation " ?" indiquent que les niveaux discrets des automates dans les chronogrammes ne sont pas encore déterminés.

Les états locaux des régulateurs de z dans ℓ correspondent à leur niveau discret pendant la période d'activation de la transition locale temporisée candidate recherchée. Cette période est entre l'instant auquel le composant ciblé change effectivement (en occurrence ici à $t = 3$ pour z) et le dernier changement apparu de tous les composants impliqués dans la transition candidate (en occurrence ici de z et des automates appartenant à ℓ).

Dans le cas général, si le changement de niveau d'un composant x est apparu à un instant i, il faut chercher le dernier instant j tel que $j < i$ avec un autre changement qui s'est apparu à j tel que ce changement est celui de x ou de l'un de ses régulateurs participant à la transition locale temporisée recherchée (noté ℓ). C'est ce que décrit l'étape 1 de l'algorithme 1, `MoT-AN` en page 93. Nous donnons dans la suite plus d'exemples sur le processus du calcul des délais des transitions locales temporisées apprises.

Remarque. On suppose qu'à l'instant initial $t = 0$, tous les composants du système sont en activité et qu'à ce moment, ils commencent tous à changer leurs niveaux.

Revenons à l'apprentissage de la transition locale temporisée qui change z du niveau 0 au niveau 1 et qui est apparue à l'instant $t_1 = 3$. Puisque ce changement est le premier changement apparu dans les chronogrammes (figure 4.8 ci-dessus) alors on considère que la transition locale temporisée qui le cause a commencé son activité à $t = 0$ (selon l'hypothèse de la remarque ci-dessus).

Les états locaux des régulateurs de z, c'est-à-dire des automates de $N_z = \{a, b\}$, sont trouvés comme suit à partir de leurs niveaux discrets dans les chronogrammes :

$$a \in N_z \Rightarrow \forall t \in [0, 3],\ \Gamma_a(t) = 0 \qquad\qquad b \in N_z \Rightarrow \forall t \in [0, 3],\ \Gamma_b(t) = 1.$$

Ainsi, a_0 et b_1 sont les états locaux respectifs de a et b et qui sont potentiellement responsables du changement de z à $t_1 = 3$ de z_0 vers z_1.

Par conséquent, pour la transition locale temporisée candidate $\tau = z_0 \xrightarrow[\delta]{\ell} z_1$, on a trois combinaisons possibles qui sont les suivantes : ℓ=$\{a_0, b_1\}$ ou ℓ=$\{a_0\}$ ou ℓ=$\{b_1\}$. Donc, il y a trois transitions locales temporisées qui sont candidates et qui diffèrent de leur condition ℓ. Étant donné qu'il s'agit du premier changement dans les chronogrammes du système et qu'on suppose qu'à $t = 0$ tous les composants commencent à changer (remarque ci-dessus), alors les délais de toutes ces transitions locales temporisées sont égaux : $\delta = 3 - 0 = 3$.

$\mathcal{T}_{change(3)}$, l'ensemble des transitions locales temporisées candidates d'être à l'origine de ce changement à $t = 3$, est alors égal à :

$$\mathcal{T}_{change(3)} = \{\tau_1 = z_0 \xrightarrow[3]{\{a_0\}} z_1, \tau_2 = z_0 \xrightarrow[3]{\{b_1\}} z_1, \tau_3 = z_0 \xrightarrow[3]{\{a_0, b_1\}} z_1\}.$$

Le deuxième changement dans la dynamique du système illustré par les chronogrammes de la figure 4.8 en page 96, se produit à $t_2 = 4$ et noté par "$change(4)$". On peut voir qu'à cet instant, c'est a qui change du niveau 0 à 1. La transition locale temporisée τ responsable de ce changement a donc cette forme : $\tau = a_0 \xrightarrow[\delta]{\ell} a_1$ où ℓ est une des combinaisons possibles des régulateurs de a et $\delta \in \mathbb{N}$ le délai correspondant. Selon le graphe d'influences $\chi(N, E)$ (figure 4.3 en page 89), $\chi_a = \{(a, a), (z, a)\}$. Donc, a peut s'auto-réguler tout seul et ainsi $\mathsf{cond}(\tau) = \ell = \emptyset$ ou bien c'est z qui le régule et dans ce cas $\ell = \{z_?\}$. Si $\mathsf{cond}(\tau) = \ell = \emptyset$, alors $\mathsf{delai}(\tau) = \delta = t_2 - t = 4 - 0 = 4$, avec $t = 0$ est l'instant où a commence son changement et $\tau = a_0 \xrightarrow[4]{\emptyset} a_1$.

Il existe d'autres transitions locales temporisées qui ont déjà été trouvées dans $\mathcal{T}_{change(3)}$, c'est-à-dire à un instant inférieur à $t_2 = 4$, et qui nécessitent que le composant a soit au niveau 0 entre $t = 0$ et $t = 3$: $z_0 \xrightarrow[3]{\{a_0\}} z_1$ et $z_0 \xrightarrow[3]{\{a_0,b_1\}} z_1$. Alors l'une de ces transitions pourrait être la cause d'un blocage de $\tau = a_0 \xrightarrow[4]{\emptyset} a_1$ pendant leur activation parce qu'elles partagent avec elle la même ressource a. En effet, elles sont considérées en conflit avec τ (définition 3.8 en page 68 du chapitre 3). Dans ce cas, la transition locale temporisée τ recherchée qui cause le changement de a à l'instant $t = 4$ ne sera débloquée qu'à l'instant oú celle qui la bloque se termine (i.e., à $t = 3$). Par la suite, $\mathsf{delai}(\tau) = \delta = 4 - 3 = 1$ et donc $\tau = a_0 \xrightarrow[1]{\emptyset} a_1$.

En revanche, si $\ell = \{z_?\}$, alors il faut voir le niveau de z entre $t = 4$ et l'instant ayant la valeur maximale entre les instants des derniers changements de a et de z avant d'atteindre $t = 4$; c'est-à-dire c'est égal à $max(\Pi^a_4, \Pi^z_4) = max(0, 3) = 3$. On a alors selon les chronogrammes, $\forall t \in [3, 4]$, $\Gamma_z(t) = 1$. Donc, $\mathsf{delai}(\tau) = \delta = 4 - 3 = 1 : \tau = a_0 \xrightarrow[1]{\{z_1\}} a_1$. Par conséquent, il y a trois transitions locales temporisées qui sont candidates pour être responsables de ce changement à $t = 4$:

$$\mathcal{T}_{change(4)} = \{\tau_1 = a_0 \xrightarrow[4]{\emptyset} a_1, \tau_2 = a_0 \xrightarrow[1]{\emptyset} a_1, \tau_3 = a_0 \xrightarrow[1]{\{z_1\}} a_1\}.$$

Nous répétons ainsi les mêmes étapes pour chaque changement produit dans les chronogrammes pour trouver les transitions locales temporisées qui sont candidates à en être responsables.

- À $t = 5$, c'est aussi a qui change de niveau mais cette fois-ci de a_1 vers a_0 avec $N_a = \{a, z\}$. Les transitions locales temporisées candidates sont alors : $\tau_1 = a_1 \xrightarrow[\delta_1]{\emptyset} a_0$ et $\tau_2 = a_1 \xrightarrow[\delta_2]{z_?} a_0$. Le dernier changement de a avant d'atteindre $t = 5$ est égal à $\Pi^a_5 = 4$ et celui de z est égal à $\Pi^z_5 = 3$. On a alors, $max(\Pi^a_5, \Pi^z_5) = max(4, 3) = 4$ donc $\delta_1 = \delta_2 = 4-3 = 1$. Et puisque $\forall t \in [4, 5]$, $\Gamma_z(t) = 1$ on a alors :

$$\mathcal{T}_{change(5)} = \{\tau_1 = a_1 \xrightarrow[1]{\emptyset} a_0, \tau_2 = a_1 \xrightarrow[1]{\{z_1\}} a_0\}.$$

- À $t = 9$, a change encore son niveau discret pour passer de 0 à 1. Les transitions locales temporisées candidates sont alors de la forme : $\tau = a_0 \xrightarrow[\delta]{\ell} a_1$. Selon les chronogrammes, entre les deux changements successifs de a, il n'existe aucun changement ni de composants régulés par a (en occurrence ici c'est b) ni de composants régulateurs de a (en occurrence ici c'est z). Donc pour toutes les transitions locales temporisées candidates, le délai ne dépend que de a : $\delta = 9 - 5 = 4$. On a alors :

$$\mathcal{T}_{change(9)} = \{\tau_1 = a_0 \xrightarrow[4]{\emptyset} a_1, \tau_2 = a_0 \xrightarrow[4]{\{z_1\}} a_1\}.$$

On peut remarquer ici que les ensembles des transitions locales temporisées candidates $\mathcal{T}_{change(9)}$ et $\mathcal{T}_{change(4)}$, qui correspondent au même changement de a (de a_0 vers a_1) ont une transition commune qui est $\tau_1 = a_0 \xrightarrow[4]{\emptyset} a_1$. On peut en déduire alors qu'il est plus probable que cette transition soit correcte puisqu'elle est retrouvée plusieurs fois pour le même changement et à des instants différents. C'est en effet l'un des filtres que nous

proposons dans la section 4.4 suivante (définition 4.5 en page 113) et qui garantit de ne garder que les transitions trouvées plus fréquemment.

- À $t = 12$, b change son état local et passe de b_1 vers b_0. On a $N_b = a$, donc a est le seul régulateur de b. Le délai δ de la transition $b_1 \xrightarrow[\delta]{\ell} b_0$ recherchée est égal à la différence entre 12 et $max(\Pi^b_{12}, \Pi^b_{12}) = max(0, 9) = 9$. Ainsi, $\delta = 12 - 9 = 3$ et on a une seule transition locale temporisée qui est candidate :

$$\mathcal{T}_{change(12)} = \{\tau_1 = b_1 \overset{\{a_1\}}{\underset{3}{\rightarrow}} b_0\}.$$

- Le changement suivant est à $t = 13$ et c'est a qui change de a_1 vers a_0. b est un composant régulé par a et son dernier changement s'est produit à l'instant $\Pi^b_{13} = 12$. z est un régulateur de a et son dernier changement s'est produit à l'instant $\Pi^z_{13} = 3 < 12$. Et on a pour a, $\Pi^a_{13} = 9 < 12$. Ainsi, la valeur maximale entre les instants des derniers changements des régulateurs de a et des composants régulés par a est égal à : $max(\Pi^a_{13}, \Pi^z_{13}, \Pi^b_{13}) = 12$. Alors il est possible que la transition recherchée responsable du changement de a à $t = 13$ était bloquée en cours de l'activation de la transition responsable du changement de b par a à $t = 12$. En effet, elles sont en conflit parce qu'elles partagent une même ressource qui est le composant a. Par conséquent, la transition locale temporisée recherchée n'aurait pu commencer son activité qu'après son déblocage à $t = 12$. Ainsi, son délai serait égal à $13 - 12 = 1$.

Les transitions locales temporisées candidates pour être responsables du changement à $t = 13$ sont présentées ci-dessous dans $\mathcal{T}_{change(13)}$. On note que les délais de τ_1 et de τ_2 sont calculés en prenant en considération le fait qu'il pourrait exister un blocage entre elles et $b_1 \overset{\{a_1\}}{\underset{3}{\rightarrow}} b_0$ de $\mathcal{T}_{change(12)}$. Mais pour les autres transitions le délai est calculé selon le dernier changement de a ou le dernier changement de z et de a si z y participe aussi.

$$\mathcal{T}_{change(13)} = \{\tau_1 = a_1 \overset{\emptyset}{\underset{1}{\rightarrow}} a_0, \tau_2 = a_1 \overset{\{z_1\}}{\underset{1}{\rightarrow}} a_0, \tau_3 = a_1 \overset{\emptyset}{\underset{4}{\rightarrow}} a_0, \tau_4 = a_1 \overset{\{z_1\}}{\underset{4}{\rightarrow}} a_0\}.$$

La même démarche est suivie pour les derniers changements à $t = 14$ et à $t = 17$:

$$\mathcal{T}_{change(14)} = \{\tau_1 = z_1 \overset{\{a_0\}}{\underset{1}{\rightarrow}} z_0, \tau_2 = z_1 \overset{\{b_0\}}{\underset{2}{\rightarrow}} z_0, \tau_3 = z_1 \overset{\{a_0, b_0\}}{\underset{1}{\rightarrow}} z_0\}.$$

$$\mathcal{T}_{change(17)} = \{\tau_1 = a_0 \overset{\emptyset}{\underset{4}{\rightarrow}} a_1, \tau_2 = a_0 \overset{\{z_0\}}{\underset{3}{\rightarrow}} a_1\}.$$

4.3.3 Génération des modèles T-AN appris

Pour générer les T-AN appris, nous procédons comme c'est indiqué à l'étape 2 de l'algorithme 1 de la méthode `Mot-AN`, en page 93. Il s'agit de calculer toutes les combinaisons possibles des transitions locales temporisées parmi celles qui sont apprises à l'étape 1 de l'algorithme 1. Chaque combinaison de transitions représente un ensemble de transitions d'un T-AN appris donné. Nous rappelons que chacune de ces combinaisons de transitions doit être minimale ayant des transitions cohérentes entre elles.

Minimalité des ensembles de transitions locales temporisées appris

Nous notons le k^{ème} T-AN appris $\mathcal{AN}^k = (\Sigma, \mathcal{S}, \mathcal{T}^k_{appris})$ avec $k \in \{0, ..., m\}$ et $m \in \mathbb{N}$ le nombre total des T-AN appris par MoT-AN. Dans la propriété 4.2 ci-dessous, nous formalisons le fait que les T-AN appris sont minimaux en expliquant toute la dynamique du système figurée dans par les chronogrammes. En effet, pour chaque changement produit dans les chronogrammes, il existe dans le T-AN appris, exactement une transition locale temporisée qui l'explique.

Propriété 4.2. *Si $\mathcal{AN}^k_{appris} = (\Sigma, \mathcal{S}, \mathcal{T}^k_{appris})$ est un T-AN appris à partir des chronogrammes Γ et du graphe d'influences χ entre les automates de $\mathcal{AN}$, alors pour tout changement d'état local d'un composant visualisé dans Γ, $\exists! \tau \in \mathcal{T}^k_{appris}$ qui réalise ce changement.*

Ainsi, la minimalité de l'ensemble de transitions locales temporisées $\mathcal{T}^k_{appris}$ est vérifiée quand pour chaque changement observé dans les chronogrammes, il y a une et une seule transition locale temporisée candidate d'en être responsable. En l'occurrence, pour l'exemple précédent (notre cas d'étude l'exemple jouet), si nous sélectionnons une et une seule transition locale temporisée de chacun des ensembles calculés $\mathcal{T}_{change(i)}$ avec $i \in \{3, 4, 5, 9, 12, 13, 14, 17\}$, alors nous aurons un $\mathcal{T}^k_{appris}$. En revanche, ceci n'est pas toujours suffisant pour vérifier complètement la minimalité d'un $\mathcal{T}^k_{appris}$. En effet, dans le cas où un même changement d'un même automate se produit dans les chronogrammes mais à des instants différents i et j avec $i \neq j$, il est possible d'apprendre des transitions identiques et donc, on a $\mathcal{T}_{change(i)} \cap \mathcal{T}_{change(j)} \neq \emptyset$.

Comme à $t = 4$ et $t = 9$ dans les chronogrammes de notre exemple jouet (figure 4.8 en page 96) où on a le même changement qui s'est produit pour le composant a (de a_0 vers a_1) et on a une même transition qui est apprise pour les deux instants : $\mathcal{T}_{change(4)} \cap \mathcal{T}_{change(9)} = \{a_0 \xrightarrow[4]{\emptyset} a_1\}$.

Pour générer un $\mathcal{T}^k_{appris}$ d'un T-AN appris, il faut sélectionner une transition de chaque $\mathcal{T}_{change(i)}$. Récapitulons ci-dessous l'ensemble des transitions locales temporisées apprises pour chaque changement observé dans les chronogrammes de l'exemple jouet dans la figure 4.8 en page 96 :

- $\mathcal{T}_{change(3)} = \{\tau_1 = z_0 \xrightarrow[3]{\{a_0\}} z_1, \tau_2 = z_0 \xrightarrow[3]{\{b_1\}} z_1, \tau_3 = z_0 \xrightarrow[3]{\{a_0, b_1\}} z_1\}$.
- $\mathcal{T}_{change(4)} = \{\tau_1 = a_0 \xrightarrow[4]{\emptyset} a_1, \tau_2 = a_0 \xrightarrow[1]{\emptyset} a_1, \tau_3 = a_0 \xrightarrow[1]{\{z_1\}} a_1\}$.
- $\mathcal{T}_{change(5)} = \{\tau_1 = a_1 \xrightarrow[1]{\emptyset} a_0, \tau_2 = a_1 \xrightarrow[1]{\{z_1\}} a_0\}$.
- $\mathcal{T}_{change(9)} = \{\tau_1 = a_0 \xrightarrow[4]{\emptyset} a_1, \tau_2 = a_0 \xrightarrow[4]{\{z_1\}} a_1\}$.
- $\mathcal{T}_{change(12)} = \{\tau_1 = b_1 \xrightarrow[3]{\{a_1\}} b_0\}$.
- $\mathcal{T}_{change(13)} = \{\tau_1 = a_1 \xrightarrow[1]{\emptyset} a_0, \tau_2 = a_1 \xrightarrow[1]{\{z_1\}} a_0, \tau_3 = a_1 \xrightarrow[4]{\emptyset} a_0, \tau_4 = a_1 \xrightarrow[4]{\{z_1\}} a_0\}$.
- $\mathcal{T}_{change(14)} = \{\tau_1 = z_1 \xrightarrow[1]{\{a_0\}} z_0, \tau_2 = z_1 \xrightarrow[2]{\{b_0\}} z_0, \tau_3 = z_1 \xrightarrow[1]{\{a_0, b_0\}} z_0\}$.
- $\mathcal{T}_{change(17)} = \{\tau_1 = a_0 \xrightarrow[4]{\emptyset} a_1, \tau_2 = a_0 \xrightarrow[3]{\{z_1\}} a_1\}$.

Pour cet exemple, si $a_0 \xrightarrow[4]{\emptyset} a_1$ est sélectionnée de $\mathcal{T}_{change(4)}$ et $a_0 \xrightarrow[4]{\{z_1\}} a_1$ est sélectionnée de $\mathcal{T}_{change(9)}$, alors on a $\mathcal{T}^k_{appris} \cap \mathcal{T}_{change(9)} = \{a_0 \xrightarrow[4]{\{z_1\}} a_1, a_0 \xrightarrow[4]{\emptyset} a_1\}$ et donc

$|\mathcal{T}^k_{appris} \cap \mathcal{T}_{change(9)}| = 2 \neq 1$. Or, pour considérer qu'un $\mathcal{T}^k_{appris}$ est minimal, il faut que pour tout changement t, $|\mathcal{T}^k_{appris} \cap \mathcal{T}_{change(t)}| = 1$ (voir algorithme 1 en page 93).

En effet, le $\mathcal{T}^k_{appris}$ n'est pas minimal dans ce cas car il existe un $\mathcal{T}^{k'}_{appris} \subset \mathcal{T}^k_{appris}$ qui peut réaliser la dynamique des chronogrammes, tel que $\mathcal{T}^{k'}_{appris} = \mathcal{T}^k_{appris} \setminus \{a_0 \xrightarrow[4]{\{z_1\}} a_1\}$.

Ainsi, pour garantir la minimalité de ce $\mathcal{T}^k_{appris}$, si c'est $a_0 \xrightarrow[4]{\emptyset} a_1$ qui est sélectionnée de $\mathcal{T}_{change(4)}$ alors elle doit être aussi sélectionnée de $\mathcal{T}_{change(9)}$.

Par contre, si c'est $a_0 \xrightarrow[1]{\{z_1\}} a_1$ qui est sélectionnée de $\mathcal{T}_{change(4)}$ alors $a_0 \xrightarrow[4]{\{z_1\}} a_1$ peut être sélectionnée de $\mathcal{T}_{change(9)}$. Et dans ce cas, la minimalité de $\mathcal{T}^k_{appris}$ est vérifiée car il y a une seule transition qui est sélectionnée de chacun de $\mathcal{T}_{change(4)}$ et $\mathcal{T}_{change(9)}$.

En résumé, si une transition τ est apprise dans plusieurs $\mathcal{T}_{change(i)}$ et si τ est selectionnée dans un $\mathcal{T}^k_{appris}$, alors pour que ce dernier soit minimal, il faut que τ soit sélectionnée dans chacun des $\mathcal{T}_{change(i)}$ où elle est apprise et qu'aucune autre transition ne soit sélectionnée de ces $\mathcal{T}_{change(i)}$ dans $\mathcal{T}^k_{appris}$.

Propriété 4.3. *Soit C l'ensemble des instants auxquels les composants du système changent d'un niveau discret vers un autre dans les chronogrammes.*

Soient $\mathcal{AN}^k_{appris} = (\Sigma, \mathcal{S}, \mathcal{T}^k_{appris})$ le $k^{ème}$ T-AN appris par la méthode d'apprentissage `MoT-AN` *avec $k \in \{1, ..m\}$, m le nombre total des T-AN appris et $\exists i, j \in C$, $i \neq j$.*

Soit $T_{ij} = \mathcal{T}_{change(i)} \cap \mathcal{T}_{change(j)}$ tel que $P \neq \emptyset$.

$$si\ \exists \tau \in T_{ij} \cap \mathcal{T}^k_{appris} \Rightarrow \mathcal{T}^k_{appris} \cap \mathcal{T}_{change(i)} = \mathcal{T}^k_{appris} \cap \mathcal{T}_{change(j)} = \tau.$$

Il est à noter que dans certains cas, la condition de minimalité peut éliminer complètement des transitions apprises telles que ces dernières n'appartiendront pas à aucun modèle appris. Ceci arrive quand il y a le même changement qui se produit à des instants différents i et j avec $i \neq j$ tel que $\mathcal{T}_{change(i)} \subset \mathcal{T}_{change(j)}$. Autrement dit, toutes les transitions qui sont apprises dans $\mathcal{T}_{change(i)}$ sont aussi apprises dans $\mathcal{T}_{change(j)}$. Ainsi, toutes les transitions de l'ensemble $\mathcal{T}_{change(j)} \setminus \mathcal{T}_{change(i)}$ n'appartiendront pas à aucun modèle appris.

Pour notre exemple, c'est le cas des transitions $a_1 \xrightarrow[1]{\emptyset} a_0$ et $a_1 \xrightarrow[1]{\{z_1\}} a_0$ dans $\mathcal{T}_{change(13)}$. On trouve que ces mêmes transitions sont apprises dans $\mathcal{T}_{change(5)}$ telles que $\mathcal{T}_{change(5)} \cap \mathcal{T}_{change(13)} = \{a_1 \xrightarrow[1]{\emptyset} a_0, a_1 \xrightarrow[1]{\{z_1\}} a_0, \} = \mathcal{T}_{change(5)}$.

Si dans un $\mathcal{T}^k_{appris}$, il y a une transition τ_1 qui est sélectionnée de $\mathcal{T}_{change(13)} \setminus \mathcal{T}_{change(5)}$ alors il y a surement une autre τ_2 qui est différente ($\tau_1 \neq \tau_2$) et qui est sélectionnée de $\mathcal{T}_{change(5)}$. Or, si $\tau \in \mathcal{T}_{change(5)}$, alors $\tau \in \mathcal{T}_{change(13)}$. Par conséquent, on a $|\mathcal{T}^k_{appris} \cap \mathcal{T}_{change(13)}| = |\tau_1, \tau_2| = 2$. Puisque pour vérifier la minimalité de $\mathcal{T}^k_{appris}$, il faut que $\mathcal{T}^k_{appris} \cap \mathcal{T}_{change(13)} = \mathcal{T}^k_{appris} \cap \mathcal{T}_{change(5)}$, ainsi les transitions de $\mathcal{T}_{change(13)} \setminus \mathcal{T}_{change(5)}$ sont toutes éliminées de l'ensemble des transitions apprises.

Propriété 4.4. *Soit C l'ensemble des instants auxquels les composants du système changent d'un niveau discret vers un autre dans les chronogrammes.*

Soient $\mathcal{AN}^k_{appris} = (\Sigma, \mathcal{S}, \mathcal{T}^k_{appris})$ le $k^{ème}$ T-AN appris par la méthode d'apprentissage `MoT-AN` *avec $k \in \{1, ..m\}$, m le nombre total des T-AN appris et $\exists i, j \in C$, $i \neq j$.*

$$\mathcal{T}_{change(i)} \subset \mathcal{T}_{change(j)} \Rightarrow \forall \mathcal{AN}^k_{appris}, (\mathcal{T}_{change(j)} \setminus \mathcal{T}_{change(i)}) \cap \mathcal{T}^k_{appris} = \emptyset.$$

Cohérence entre les transitions locales temporisées du même T-AN

Toutes les transitions apprises appartenant à un même $\mathcal{T}^k_{appris}$ doivent être *cohérentes entre elles*. Ce que nous voulons dire par cohérentes entre elles c'est que, s'il existe une transition locale temporisée ayant un délai calculé selon un blocage dû à un conflit avec une autre transition locale temporisée alors ces deux transitions doivent appartenir au même $\mathcal{T}^k_{appris}$. Ainsi, une fois simulées, elles doivent pouvoir re-réaliser les chronogrammes à partir desquels elles sont apprises et selon la même sémantique de la dynamique des T-AN.

Par exemple, les transitions locales temporisées $z_0 \overset{\{a_0,b_1\}}{\underset{3}{\rightarrow}} z_1$ et $z_0 \overset{\{a_0\}}{\underset{3}{\rightarrow}} z_1$ dans $\mathcal{T}_{change(3)}$ sont en conflit avec $a_0 \overset{\emptyset}{\underset{1}{\rightarrow}} a_1$ dans $\mathcal{T}_{change(4)}$ car elles partagent la même ressource a. Principalement ce conflit est visualisé par le fait que le délai qui est égal à 1 dans $a_0 \overset{\emptyset}{\underset{1}{\rightarrow}} a_1$ est calculé en considérant ce conflit. Ainsi, il est possible d'avoir par exemple, dans un même $\mathcal{T}^k_{appris}$ la transition $z_0 \overset{\{a_0,b_1\}}{\underset{3}{\rightarrow}} z_1$ (ou $z_0 \overset{\{a_0\}}{\underset{3}{\rightarrow}} z_1$) de $\mathcal{T}_{change(3)}$ et $a_0 \overset{\emptyset}{\underset{1}{\rightarrow}} a_1$ de $\mathcal{T}_{change(4)}$. Et dans un autre même $\mathcal{T}^p_{appris}$, $z_0 \overset{\{b_1\}}{\underset{3}{\rightarrow}} z_1$ de $\mathcal{T}_{change(3)}$ et $a_0 \overset{\emptyset}{\underset{4}{\rightarrow}} a_1$ de $\mathcal{T}_{change(4)}$.

En revanche, les transitions $z_0 \overset{\{b_1\}}{\underset{3}{\rightarrow}} z_1$ de $\mathcal{T}_{change(3)}$ et $a_0 \overset{\emptyset}{\underset{1}{\rightarrow}} a_1$ de $\mathcal{T}_{change(4)}$ ne peuvent pas appartenir au même modèle. En effet, le délai de la deuxième transition est égal à 1 car nous avons tenu compte du fait qu'il existe dans le même modèle une autre transition locale temporisée qui est en conflit avec elle. Sinon, son délai devrait être égal à 4.

Nous avons aussi les transitions $a_1 \overset{\emptyset}{\underset{1}{\rightarrow}} a_0$ et $a_1 \overset{\{z_1\}}{\underset{1}{\rightarrow}} a_0$ de $\mathcal{T}_{change(13)}$ dont le délai est calculé parce qu'il y a un conflit avec la transition $b_1 \overset{\{a_1\}}{\underset{3}{\rightarrow}} b_0$ de $\mathcal{T}_{change(12)}$. Ainsi, dans un même modèle, on pourrait avoir $a_1 \overset{\emptyset}{\underset{1}{\rightarrow}} a_0$ (ou $a_1 \overset{\{z_1\}}{\underset{1}{\rightarrow}} a_0$) et $b_1 \overset{\{a_1\}}{\underset{3}{\rightarrow}} b_0$.

Résultat de MoT-AN

Même avec les contraintes de minimalité et de cohérence introduite ci-dessous sur les ensembles des transitions locales temporisées apprises par modèle, plusieurs combinaisons sont possibles des transitions et donc plusieurs modèles T-AN peuvent être appris. Le nombre total de ces modèles appris $m \in \mathbb{N}$ est borné par le produit des nombres des transitions apprises pour chaque changement observé dans les chronogrammes (voir définition 4.5 ci-dessous). En effet, quand nous générons un modèle T-AN, nous sélectionnons pour chaque changement observé à un instant t au plus une transition pour l'expliquer à partir des transitions apprises dans $\mathcal{T}_{change(t)}$.

Propriété 4.5 (Nombre total des modèles T-AN appris)**.** *Soit C l'ensemble des instants auxquels les composants du système changent d'un niveau discret vers un autre dans les chronogrammes illustrant la dynamique du système modélisé. Le nombre total des changements est alors égal à $|C|$.*

Soit $m \in \mathbb{N}$ le nombre total des modèles T-AN appris par l'algorithme 1, Mot-AN, on a donc,

$$m \leq \prod_{i=1}^{|C|} |\mathcal{T}_{change(i)}|$$

et si $\exists i, j \in C,\ i \neq j$ *tel que* $\mathcal{T}_{change(i)} \cap \mathcal{T}_{change(j)} \neq \emptyset$ *alors,*

$$m \leq \prod_{i=1}^{|C|} |\mathcal{T}_{change(i)} \setminus \bigcup_{j<i} \mathcal{T}_{change(j)}|$$

En effet, m est aussi borné par le cardinal du produit cartésien des $|C|$ ensembles $\mathcal{T}_{change(i)}$ privés des éléments existants en commun dans les autres ensembles, c'est-à-dire privés de $\bigcup_{j<i} \mathcal{T}_{change(j)}$. Autrement dit, l'élément en commun sera pris seulement du premier ensemble où il est apparu. En effet, puisqu'on exige que de chaque $\mathcal{T}_{change(j)}$ nous sélectionnons une unique transition et il ne faut pas tenir compte de ces répétitions dans le calcul du nombre des combinaisons possibles.

Pour notre exemple jouet étudié et dont les chronogrammes sont présentés dans la figure 4.8 en page 96, l'ensemble des instants auxquels produisent les changements dans les chronogrammes, C, est alors égal à :
$C = \{\ c_1^a = 4,\ c_2^a = 5,\ c_3^a = 9,\ c_4^a = 13,\ c_5^a = 17,\ c_1^b = 12,\ c_1^z = 3,\ c_2^z = 14\ \}$ avec c_j^x est le changement d'indice j du composant x. Il y a donc $|C| = 8$ changements au total dans les chronogrammes.

En prenant le récapitulatif en page 100 des transitions locales temporisée apprises pour chaque changement apparu dans les chronogrammes, nous pouvons déduire que :

- $\mathcal{T}_{change(3)} \cap \bigcup_{j<3} \mathcal{T}_{change(j)} = \mathcal{T}_{change(4)} \cap \bigcup_{j<4} \mathcal{T}_{change(j)} = \mathcal{T}_{change(5)} \cap \bigcup_{j<5} \mathcal{T}_{change(j)} = \emptyset$,
- $\mathcal{T}_{change(9)} \cap \bigcup_{j<9} \mathcal{T}_{change(j)} = \mathcal{T}_{change(9)} \cap \mathcal{T}_{change(4)} = \{a_0 \xrightarrow[4]{\emptyset} a_1\}$,
- $\mathcal{T}_{change(12)} \cap \bigcup_{j<12} \mathcal{T}_{change(j)} = \emptyset$,
- $\mathcal{T}_{change(13)} \cap \bigcup_{j<13} \mathcal{T}_{change(j)} = \mathcal{T}_{change(13)} \cap \mathcal{T}_{change(5)} = \{a_1 \xrightarrow[1]{\emptyset} a_0, a_1 \xrightarrow[1]{\{z_1\}} a_0\}$,
- $\mathcal{T}_{change(14)} \cap \bigcup_{j<14} \mathcal{T}_{change(j)} = \emptyset$,
- $\mathcal{T}_{change(17)} \cap \bigcup_{j<17} \mathcal{T}_{change(j)} = \mathcal{T}_{change(17)} \cap (\mathcal{T}_{change(4)} \cup \mathcal{T}_{change(9)}) = \{a_0 \xrightarrow[4]{\emptyset} a_1\}$.

On a alors :

- $|\mathcal{T}_{change(3)} \setminus \bigcup_{j<3} \mathcal{T}_{change(j)}| = |\mathcal{T}_{change(3)}| = 3$
- $|\mathcal{T}_{change(4)} \setminus \bigcup_{j<4} \mathcal{T}_{change(j)}| = |\mathcal{T}_{change(4)}| = 3$
- $|\mathcal{T}_{change(5)} \setminus \bigcup_{j<5} \mathcal{T}_{change(j)}| = |\mathcal{T}_{change(5)}| = 2$
- $|\mathcal{T}_{change(9)} \setminus \bigcup_{j<9} \mathcal{T}_{change(j)}| = |\{a_0 \xrightarrow[4]{\{z_1\}} a_1\}| = 1$
- $|\mathcal{T}_{change(12)} \setminus \bigcup_{j<12} \mathcal{T}_{change(j)}| = |\mathcal{T}_{change(12)}| = 1$
- $|\mathcal{T}_{change(13)} \setminus \bigcup_{j<13} \mathcal{T}_{change(j)}| = |\{a_1 \xrightarrow[4]{\emptyset} a_0, a_1 \xrightarrow[4]{\{z_1\}} a_0\}| = 2$
- $|\mathcal{T}_{change(14)} \setminus \bigcup_{j<14} \mathcal{T}_{change(j)}| = |\mathcal{T}_{change(14)}| = 3$
- $|\mathcal{T}_{change(17)} \setminus \bigcup_{j<17} \mathcal{T}_{change(j)}| = |\{a_0 \xrightarrow[3]{\{z_1\}} a_1\}| = 1$

Ainsi, $m \in \mathbb{N}$ le nombre total des modèles T-AN appris par MoT-AN est borné par : $m \leq \prod_{i \in C} |\mathcal{T}_{change(i)} \setminus \bigcup_{j<i} \mathcal{T}_{change(j)}| = 3 \times 3 \times 2 \times 1 \times 1 \times 2 \times 3 \times 1 = 108$ modèles.

Nous notons que ce nombre est souvent non atteint. En effet, il y a des transitions locales temporisées qui ne peuvent pas appartenir à un même modèle T-AN appris ; quand il s'agit d'un ensemble de transitions qui ne vérifie pas la condition de minimalité. Comme il a été évoqué précédemment, les transitions $a_1 \xrightarrow[4]{\emptyset} a_0$ et $a_1 \xrightarrow[4]{\{z_1\}} a_0$ apprises dans $\mathcal{T}_{change(13)}$ ne pourront appartenir à aucun T-AN appris car elles appartiennent à $\mathcal{T}_{change(13)} \setminus \mathcal{T}_{change(5)}$ sachant que $\mathcal{T}_{change(5)} \subset \mathcal{T}_{change(13)}$ (voir la propriété 4.4 en page 101). Ainsi, quelle que soit la transition choisie de $\mathcal{T}_{change(5)}$, elle est suffisante pour expliquer le changement produit à $t = 5$ et aussi au changement produit à $t = 13$.

Ceci réduit alors le nombre des T-AN appris car il n'y a aucune transition supplémentaire à prendre en considération de $\mathcal{T}_{change(13)}$ lors de la génération des combinaisons possibles. Autrement dit, celle de $\mathcal{T}_{change(5)}$ sont suffisantes.

Quand nous enlevons alors $|\mathcal{T}_{change(13)} \setminus \bigcup_{j<13} \mathcal{T}_{change(j)}| = 2$ du produit qui calcule la borne maximale du nombre total des T-AN appris et nous avons :
$m \leq 3 \times 3 \times 2 \times 1 \times 1 \times 3 \times 1 = 54$ modèles. On remarque que l'élimination de seulement 2 transitions parmi celles qui sont apprises pourrait diviser le nombre des modèles appris par deux.

D'autre part, la condition de la cohérence entre les transitions dans un même modèle T-AN appris réduit considérablement le nombre des T-AN appris. Rappelons que cette cohérence consiste à vérifier que des transitions locales temporisées dont les délais sont calculés en se basant sur le fait qu'elles sont en conflit doivent appartenir au même modèle.

Pour notre exemple, il y a les transitions $z_0 \xrightarrow[3]{\{a_0,b_1\}} z_1$ et $z_0 \xrightarrow[3]{\{a_0\}} z_1$ dans $\mathcal{T}_{change(3)}$ qui sont en conflit avec $a_0 \xrightarrow[1]{\emptyset} a_1$ dans $\mathcal{T}_{change(4)}$. Donc, si $a_0 \xrightarrow[1]{\emptyset} a_1$ est sélectionnée dans un modèle alors $z_0 \xrightarrow[3]{\{a_0\}} z_1$ ou $z_0 \xrightarrow[3]{\{a_0,b_1\}} z_1$ doit y être aussi. Ainsi, aucune autre transition de $\mathcal{T}_{change(3)}$ ne pourrait être dans le même modèle avec $a_0 \xrightarrow[1]{\emptyset} a_1$.

Par conséquent, il est possible de soustraire les combinaisons en trop des T-AN qui incluent $a_0 \xrightarrow[1]{\emptyset} a_1$ de $\mathcal{T}_{change(4)}$ et l'autre transition de $\mathcal{T}_{change(3)}$, c'est-à-dire $\mathcal{T}_{change(3)} \setminus \{z_0 \xrightarrow[3]{\{a_0,b_1\}} z_1, z_0 \xrightarrow[3]{\{a_0\}} z_1\} = \{z_0 \xrightarrow[3]{\{b_1\}} z_1\}$. Le nombre de combinaison à enlever $m_e \in \mathbb{N}$ est alors égal à :
$m_e = |\{z_0 \xrightarrow[3]{\{b_1\}} z_1\}| \times |\{a_0 \xrightarrow[1]{\emptyset} a_1\}| \times \prod_{i \in C \setminus \{3,4\}} |\mathcal{T}_{change(i)} \setminus \bigcup_{j<i} \mathcal{T}_{change(j)}|$
$m_e = 1 \times 1 \times 2 \times 1 \times 1 \times 1 \times 3 \times 1 = 6$.

Le nombre des modèles T-AN appris pour cet exemple jouet à partir des chronogrammes de la figure 4.8, en page 96 devient ainsi borné par $54 - 6 = 48$ modèles ($m \leq 48$).

Il est à noter aussi qu'il y a d'autres critères possibles pour réduire encore plus ce nombre de modèles appris et pour raffiner les modèles appris. Nous les présentons comme des filtres dans la section 4.4 en page 110.

Nous donnons ci-dessous quelques exemples des ensembles de transitions locales temporisées des modèles T-AN appris à partir des chronogrammes de la figure 4.8 en page 96 par la méthode MoT-AN dont l'algorithme est donné en page 93 et son implémentation

en Answer Set Programming est présentée dans le chapitre 6. Le nombre des modèles retournés est égal à 48 qui est en effet égal à la borne maximale trouvée ci-dessus.

- $\mathcal{T}^1_{appris} = \{\tau_1 = a_0 \xrightarrow[4]{\{z_1\}} a_1,\ \tau_2 = a_0 \xrightarrow[3]{\{z_0\}} a_1,\ \tau_3 = a_1 \xrightarrow[1]{\emptyset} a_0,\ \tau_4 = a_0 \xrightarrow[1]{\emptyset} a_1,$
  $\tau_5 = b_1 \xrightarrow[3]{\{a_1\}} b_0, \tau_6 = z_0 \xrightarrow[3]{\{a_0,b_1\}} z_1,\ \tau_7 = z_1 \xrightarrow[1]{\{a_0\}} z_0,\ \}.$
- $\mathcal{T}^2_{appris} = \{\tau_1 = a_0 \xrightarrow[1]{\{z_1\}} a_1,\ \tau_2 = a_1 \xrightarrow[4]{\{z_1\}} a_0,\ \tau_3 = a_0 \xrightarrow[3]{\{z_0\}} a_1,\ \tau_4 = a_0 \xrightarrow[1]{\emptyset} a_1,$
  $\tau_5 = b_1 \xrightarrow[3]{\{a_1\}} b_0, \tau_6 = z_0 \xrightarrow[3]{\{a_0\}} z_1,\ \tau_7 = z_1 \xrightarrow[1]{\{a_0,b_0\}} z_0\ \}.$
- $\mathcal{T}^3_{appris} = \{\tau_1 = a_0 \xrightarrow[4]{\emptyset} a_1,\ \tau_2 = a_1 \xrightarrow[1]{\{z_1\}} a_0,\quad \tau_3 = b_1 \xrightarrow[3]{\{a_1\}} b_0,$
  $\tau_4 = z_0 \xrightarrow[3]{\{a_0,b_1\}} z_1,\ \tau_5 = z_1 \xrightarrow[2]{\{b_0\}} z_0\ \}.$
- $\mathcal{T}^4_{appris} = \{\tau_1 = a_0 \xrightarrow[1]{\emptyset} a_1,\ \tau_2 = a_1 \xrightarrow[1]{\emptyset} a_0,\ \tau_3 = a_0 \xrightarrow[4]{\{z_1\}} a_1,\ \tau_4 = a_0 \xrightarrow[3]{\{z_0\}} a_1,$
  $\tau_5 = b_1 \xrightarrow[3]{\{a_1\}} b_0, \tau_6 = z_0 \xrightarrow[3]{\{a_0\}} z_1,\ \tau_7 = z_1 \xrightarrow[1]{\{a_0,b_0\}} z_0\ \}...$

4.3.4 Comparaison entre les T-AN appris et le T-AN initial

Nous rappelons que nous notons par $\mathcal{T}_{appris}$ l'ensemble de toutes les transitions locales temporisées apprises. Autrement dit, c'est l'union des ensembles des transitions locales temporisées de tous les modèles T-AN appris.

Propriété 4.6. *Soit $\mathcal{T}^k_{appris}$ l'ensemble des transitions locales temporisées apprises dans un modèle T-AN, tel que $k \in \{1, .., m\}$ et $m \in \mathbb{N}$ le nombre total des modèles appris par l'algorithme 1, Mot-AN, à partir d'un ensemble de chronogrammes Γ. On a alors $\mathcal{T}_{appris} = \bigcup_{k=1}^{m} \mathcal{T}^k_{appris}$ est l'ensemble de toutes les transitions locales temporisées apprises.*

Si les contraintes de minimalité d'un modèle appris et de la cohérence des transitions dans un modèle appris n'éliminent pas des transitions apprises des $\mathcal{T}_{change(i)}$ alors elles sont toutes retrouvées dans $\mathcal{T}_{appris}$. Par exemple, pour ce cas d'étude de l'exemple jouet, on a mentionné ci-dessus (dans la propriété 4.4 en page 101) que la condition de minimalité des T-AN élimine les transitions $a_1 \xrightarrow[4]{\emptyset} a_0$ et $a_1 \xrightarrow[4]{\{z_1\}} a_0$ apprises dans $\mathcal{T}_{change(13)}$.

Par conséquent, on peut considérer que $\mathcal{T}_{appris} \subseteq \bigcup_{i \in C} \mathcal{T}_{change(i)}$.

La validation pratique du résultat de l'algorithme 1, `Mot-AN`, est effectuée par la comparaison entre l'ensemble des transitions locales temporisées apprises (dans $\mathcal{T}_{appris}$) et l'ensemble des transitions locales temporisées du modèle initial (dans $\mathcal{T}_{initial}$). Nous rappelons que les transitions dans $\mathcal{T}_{appris}$ sont apprises par l'algorithme 1, `MoT-AN`, à partir des chronogrammes trouvés par la simulation de l'ensemble des transitions du modèle T-AN initial ($\mathcal{T}_{initial}$). La démarche de la comparaison est présentée dans la figure 4.6 en page 95.

Le résultat de `MoT-AN` est dit validé si on réussit à retrouver toutes les transitions du $\mathcal{T}_{initial}$ dans $\mathcal{T}_{appris}$ (i.e., $\mathcal{T}_{initial} \subseteq \mathcal{T}_{appris}$). Autrement dit, on répond toujours positivement à la question suivante : $\forall \tau \in \mathcal{T}_{initial}\ \exists ? \tau' \in \mathcal{T}_{appris}$ telle que $\tau = \tau'$?

Calculons dans la suite, l'ensemble total des transitions locales temporisées apprises pour l'exemple jouet étudié. On rappelle que selon les chronogrammes de la figure 4.8 en page 96, l'ensemble des instants auxquels les composants du T-AN changent d'un niveau discret vers un autre est : $C = \{4, 5, 9, 13, 17, 3, 14, 12, 4\}$ et que le nombre des modèles appris par MoT-AN est égal à 48.

Selon la propriété 4.6 ci-dessus, $\mathcal{T}_{appris} = \bigcup_{k=1}^{48} \mathcal{T}^k_{appris} \subseteq \bigcup_{j \in C} \mathcal{T}_{change(j)}$.

Nous savons que pour cet exemple, les deux transitions $a_1 \xrightarrow[4]{\emptyset} a_0$ et $a_1 \xrightarrow[4]{\{z_1\}} a_0\}$ sont éliminées par la condition de minimalités car $\mathcal{T}_{change(5)} \subset \mathcal{T}_{change(13)}$. Donc $\mathcal{T}_{appris} = \bigcup_{i \in C} \mathcal{T}_{change(i)} \setminus \{a_1 \xrightarrow[4]{\emptyset} a_0, a_1 \xrightarrow[4]{\{z_1\}} a_0\}$. Récapitulons alors ci-dessous $\mathcal{T}_{appris}$ sachant que $\forall i \in C$ les $\mathcal{T}_{change(i)}$ sont récapitulés en page 100.

$$
\begin{aligned}
\mathcal{T}_{appris} = \{ & \\
& \tau_1 = z_0 \xrightarrow[3]{\{a_0\}} z_1, \ \ \tau_2 = z_0 \xrightarrow[3]{\{b_1\}} z_1, \ \ \tau_3 = z_0 \xrightarrow[3]{\{a_0,b_1\}} z_1, \\
& \tau_4 = z_1 \xrightarrow[1]{\{a_0\}} z_0, \ \ \tau_5 = z_1 \xrightarrow[2]{\{b_0\}} z_0, \ \ \tau_6 = z_1 \xrightarrow[1]{\{a_0,b_0\}} z_0, \\
& \tau_7 = b_1 \xrightarrow[3]{\{a_1\}} b_0, \\
& \tau_8 = a_0 \xrightarrow[4]{\emptyset} a_1, \ \ \tau_9 = a_0 \xrightarrow[1]{\emptyset} a_1, \ \ \tau_{10} = a_0 \xrightarrow[1]{\{z_1\}} a_1, \ \ \tau_{11} = a_0 \xrightarrow[3]{\{z_0\}} a_1, \ \ \tau_{12} = a_0 \xrightarrow[4]{\{z_1\}} a_1, \\
& \tau_{13} = a_1 \xrightarrow[1]{\emptyset} a_0, \ \ \tau_{14} = a_1 \xrightarrow[1]{\{z_1\}} a_0 \\
& \}.
\end{aligned}
$$

Rappelons ci-dessous la description textuelle de l'ensemble des transitions locales temporisées $\mathcal{T}_{initial}$ du T-AN initial (c'est-à-dire de l'exemple jouet de la figure 4.7 en page 96).

$$
\begin{aligned}
\mathcal{T}_{initial} = \{ & \tau_1 = a_0 \xrightarrow[4]{\emptyset} a_1, \ \ \tau_2 = a_1 \xrightarrow[1]{\{z_1\}} a_0, \qquad \tau_3 = b_1 \xrightarrow[3]{\{a_1\}} b_0, \\
& \tau_4 = z_0 \xrightarrow[3]{\{a_0,b_1\}} z_1, \ \ \tau_5 = z_1 \xrightarrow[2]{\{b_0\}} z_0, \ \}.
\end{aligned}
$$

Vérifions si ces 5 transitions locales temporisées du $\mathcal{T}_{initial}$ existent aussi dans $\mathcal{T}_{appris}$:

- $\tau = a_0 \xrightarrow[4]{\emptyset} a_1$, $\tau = \tau_1 \in \mathcal{T}_{initial}$ et $\tau = \tau_8 \in \mathcal{T}_{appris} \Rightarrow \tau_1 \in \mathcal{T}_{appris}$;
- $\tau = a_1 \xrightarrow[1]{\{z_1\}} a_0$, $\tau = \tau_2 \in \mathcal{T}_{initial}$ et $\tau = \tau_{14} \in \mathcal{T}_{appris} \Rightarrow \tau_2 \in \mathcal{T}_{appris}$;
- $\tau = b_1 \xrightarrow[3]{\{a_1\}} b_0$, $\tau = \tau_3 \in \mathcal{T}_{initial}$ et $\tau = \tau_7 \in \mathcal{T}_{appris} \Rightarrow \tau_3 \in \mathcal{T}_{appris}$;
- $\tau = z_0 \xrightarrow[3]{\{a_0,b_1\}} z_1$, $\tau = \tau_4 \in \mathcal{T}_{initial}$ et $\tau = \tau_3 \in \mathcal{T}_{appris} \Rightarrow \tau_4 \in \mathcal{T}_{appris}$;
- $\tau = z_1 \xrightarrow[2]{\{b_0\}} z_0$, $\tau = \tau_5 \in \mathcal{T}_{initial}$ et $\tau = \tau_5 \in \mathcal{T}_{appris} \Rightarrow \tau_5 \in \mathcal{T}_{appris}$;

Donc $\forall \tau \in \mathcal{T}_{initial}$, $\exists \tau' \in \mathcal{T}_{appris}$ telle que $\tau = \tau'$ ainsi $\mathcal{T}_{initial} \subset \mathcal{T}_{appris}$.

De plus, si nous vérifions dans les modèles T-AN appris par Mot-AN, dont quelques un sont détaillés en page 105, on peut remarquer que l'ensemble des transitions locales temporisées $\mathcal{T}^3_{appris}$ est identique à $\mathcal{T}_{initial}$.

En effet, ce modèle ne contient que des transitions locales temporisées qui sont correctes c'est-à-dire celles qui sont effectivement à l'origine des changements de la dynamique du système illustrée par les chronogrammes.

Ainsi, nous avons pu mettre en exergue sur ce petit exemple, la complétude de l'algorithme 1, `Mot-AN` (théorème 4.1 en page 108). Donc, si les chronogrammes donnés comme entrée à `MoT-AN` respectent la sémantique des T-AN, introduite dans le chapitre 3 précédent —qui est la sémantique sur laquelle est basée l'algorithme d'apprentissage `MoT-AN`— et si toutes les transitions locales temporisées du T-AN initial ont été activées au moins une fois, alors il existe au moins un T-AN parmi les T-AN appris qui est identique au modèle initial.

A cause du non déterminisme dans la sémantique —déjà discuté dans la section 3.3.2 en page 66— il n'est pas possible de décider si une transition locale temporisée est absolument correcte ou non, c'est-à-dire si c'est effectivement elle qui a causé le changement d'état local d'un composant dans la dynamique du système.

En plus, comme les données de séries temporelles, qui sont issues des mesures des expériences biologiques réelles, ne sont pas parfaites (i.e., bruitées à cause par exemple des erreurs de mesures), les résultats de leurs discrétisation, (i.e., les chronogrammes) pourraient aussi contenir des données bruitées. Ainsi, le défi est alors de supprimer de $\mathcal{T}_{appris}$ les transitions locales temporisées apprises qui ne sont pas correctes. C'est sur quoi porte la section 4.4 suivante. En effet, nous affinons l'ensemble des modèles retournés de l'algorithme 1, `Mot-AN`, parce que nous faisons appel à des *filtres* et dont le but est d'éliminer le plus possible les incohérences dans les modèles T-AN retournés. Mais nous étudions d'abord, dans la sous-section suivante, la complétude de l'algorithme 1, `MoT-AN`.

4.3.5 Étude théorique de `MoT-AN`

Une question qui se pose sur les résultats de l'algorithme `MoT-AN` concerne sa *complétude* : l'algorithme garantit-il de retrouver toutes les transitions locales temporisées qui sont effectivement responsables des changements dans les chronogrammes pris comme entrée ? Nous avons pu le valider en pratique dans la section précédente en utilisant l'exemple jouet et dans cette section nous montrons aussi sa complétude théorique dans le théorème 4.1 ci-dessous.

Théorème 4.1 (Complétude)**.** *Soit $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$ un T-AN, Γ les chronogrammes des composants de $\mathcal{AN}$, n le nombre maximal des régulateurs d'un composant donné avec $n \in \mathbb{N}$ et $R \subseteq \mathcal{T}$ l'ensemble des transitions locales temporisées qui réalisent les chronogrammes Γ, tel que $\forall a_i \xrightarrow[\delta]{\ell} a_j \in R \implies |\ell| \leq n$. Soit χ le graphe d'influences entre les automates de Σ et $\mathcal{AN}' = (\Sigma, \mathcal{S}, \emptyset)$ un T-AN. $\mathcal{AN}'$, Γ, χ et n sont donnés comme entrée à l'algorithme 1, MoT-AN. Ce dernier est complet car il va générer en sortie un ensemble de T-AN Φ tel que $\exists \mathcal{AN}^k_{appris} = (\Sigma, \mathcal{S}, \mathcal{T}^k_{appris}) \in \Phi$ avec $R \subseteq \mathcal{T}^k_{appris}$ et $k \in \{1, ..., |\Phi|\}$.*

Démonstration. Supposons que l'algorithme 1 n'est pas complet, alors $\exists \tau \in R$ qui réalise un changement dans Γ telle que $\forall \mathcal{AN}^k_{appris} = (\Sigma, \mathcal{S}, \mathcal{T}^k_{appris}) \in \Phi$ avec $k \in \{1, ..., |\Phi|\}$, $\tau \notin \mathcal{T}^k_{appris}$. Après l'étape 1 de l'algorithme 1, MoT-AN, génèrent selon le graphe d'influences χ dans $\mathcal{T}_{appris}$ toutes les transitions locales temporisées qui sont candidates de pouvoir réaliser tous les changement dans Γ. Ainsi, $\nexists \tau \in R$ qui réalise effectivement un changement dans Γ qui n'est pas générée donc $\tau \in \mathcal{T}_{appris}$. Puisque τ réalise effectivement l'un des changements de Γ alors τ est générée à l'étape 1 pour tous les changements duquel elle est responsable, et donc elle ne sera pas éliminée quand MoT-AN calcule les ensembles minimaux des transitions locales temporisées des T-AN appris. Alors τ sera nécessairement présente dans au moins l'un des ensembles minimaux des transitions locales temporisées des T-AN retournés : $\exists \mathcal{AN}^k_{appris} = (\Sigma, \mathcal{S}, \mathcal{T}^k_{appris}) \in \Phi$ avec $k \in \{1, ..., |\Phi|\}$ tel que $\tau \in \mathcal{T}^k_{appris}$. □

Par conséquent, le théorème 4.1 ci-dessus montre qu'il n'existe pas un changement de niveau d'un composant observé dans les chronogrammes (pris en entrée par l'algorithme 1, MoT-AN) sans qu'il soit expliqué dans au moins l'un des T-AN appris par la transition locale temporisée correcte ; c'est-à-dire celle qui a effectivement causé le changement. Ainsi, la propriété 4.7 ci-dessous est déduite directement de ce théorème.

Propriété 4.7. *Si $\mathcal{T}_{appris}$ est l'ensemble des transitions locales temporisées apprises par l'algorithme 1, MoT-AN, à partir des chronogrammes Γ et R l'ensemble des transitions locales temporisées qui ont été activées pour avoir le chronogramme Γ alors $R \subseteq \mathcal{T}_{appris}$.*

Dans la sous-section suivante, nous appliquons la méthode la méthode d'apprentissage MoT-AN sur d'autre modèles biologiques de taille plus grande.

4.3.6 Validation pratique du résultat de MoT-AN sur d'autres exemples

Nous présentons dans la suite les résultats obtenus après l'application de l'algorithme d'apprentissage 1, MoT-AN, sur des exemples des systèmes biologiques réels. Ces systèmes sont modélisés avec le formalisme des T-AN dont la taille de chacun (i.e., le nombre de composants) est supérieur à celle de l'exemple jouet.

Ainsi, nous faisons la même démarche de validation que celle qui a été appliquée pour le T-AN de l'exemple jouet. Les étapes de la validation pratique sont détaillées dans la sous-section 4.3.1 en page 94.

Pour comparer entre les modèles qui sont appris et le modèle initial, nous prenons l'union de tous les modèles T-AN appris. Ainsi nous avons l'ensemble de toutes les transitions locales temporisées apprises. Ensuite, nous vérifions que l'ensemble des transitions locales temporisées du T-AN initial est inclus dans l'ensemble des transitions locales temporisées apprises.

Dans le tableau 4.1 ci-dessous, nous donnons les caractéristiques des modèles traités :

- Exemple jouet : le modèle T-AN de la figure 4.7 en page 96 ;
- Lambda phage : le bactériophage lambda (Thieffry & Thomas, 1995) qui présente des gènes viraux pour modéliser le réseau de régulation complexe nécessaire à la décision entre la lyse et la lysogénisation dans le bactériophage tempéré ;
- Mammalian : le cycle cellulaire de mammifères (Fauré et al., 2006).

Modèles	**Description**		
	$\|\Sigma\|$	$\|\mathcal{S}\|$	$\|\mathcal{T}_{initial}\|$
Exemple jouet	3	8	5
Lambda phage	4	48	46
Mammalian	10	2^{10}	34

Table 4.1 : Tableau récapitulatif des dimensions de quelques systèmes biologiques sur lesquels nous avons appliqué `MoT-AN`. Les lignes successives résument les informations relatives à, respectivement, l'"exemple jouet" de la figure 4.7, le "Lambda phage" (Thieffry & Thomas, 1995), et le cycle cellulaire des mammifères "Mammilian" (Fauré et al., 2006). Pour chacun de ces modèles T-AN, ce tableau donne le nombre d'automates ($|\Sigma|$), le nombre d'états globaux dans le graphe d'états correspondant ($|\mathcal{S}|$) et le nombre de transitions locales temporisées ($|\mathcal{T}_{initial}|$).

Nous avons effectué les tests de la validation pratique de `MoT-AN` sur une centaine de chronogrammes issus des simulations des modèles décrits dans le tableau 4.1. Nous avons fait varier à chaque fois la taille des chronogrammes issus des simulations. Ensuite, nous avons lancé l'algorithme 1, `MoT-AN`, sur chaque ensemble de chronogrammes d'un modèle donné. Finalement, nous avons pris l'union de tous les modèles appris et nous avons comparé l'ensemble des transitions locales temporisées apprises avec l'ensemble des transitions locales temporisées du modèle initial. La même démarche est suivie précédemment pour une seule simulation de l'exemple jouet dont les chronogrammes sont de taille égale à 18 (figure 4.8 en page 96).

Dans la figure 4.9 ci-dessous, nous détaillons les résultats de la comparaison entre les transitions locales temporisées de l'union de tous les modèles appris par l'algorithme 1, `MoT-AN` ($\mathcal{T}_{appris}$), et celles du modèle initial ($\mathcal{T}_{initial}$) en variant la taille maximale des chronogrammes pris en entrée.

Nous remarquons que plus la taille des chronogrammes est importante (l'axe des abscisses), plus le nombre des transitions locales temporisées apprises augmente (les deux barres vertes et bleues). Ainsi, le nombre des transitions locales temporisées qui sont correctes (c'est-à-dire qui existent aussi dans le T-AN initial) présentées par les barres vertes sur la figure 4.9 augmente jusqu'à atteindre le nombre maximal des transitions locales temporisées du T-AN initial (présenté par la ligne discontinue rouge pour chaque modèle sur la figure 4.9).

Le fait de simuler un T-AN sur un certain nombre d'instants ne nous permet pas de nous assurer d'avoir activé toutes ses transitions locales temporisées. Nous rappelons que si une transition locale temporisée est activée, alors elle cause un changement d'état local d'un automate du réseau. Donc, l'algorithme 1, `Mot-AN`, va l'apprendre (voir théorème 4.1 en page 108 concernant la complétude de `MoT-AN`). Ainsi, si une transition locale

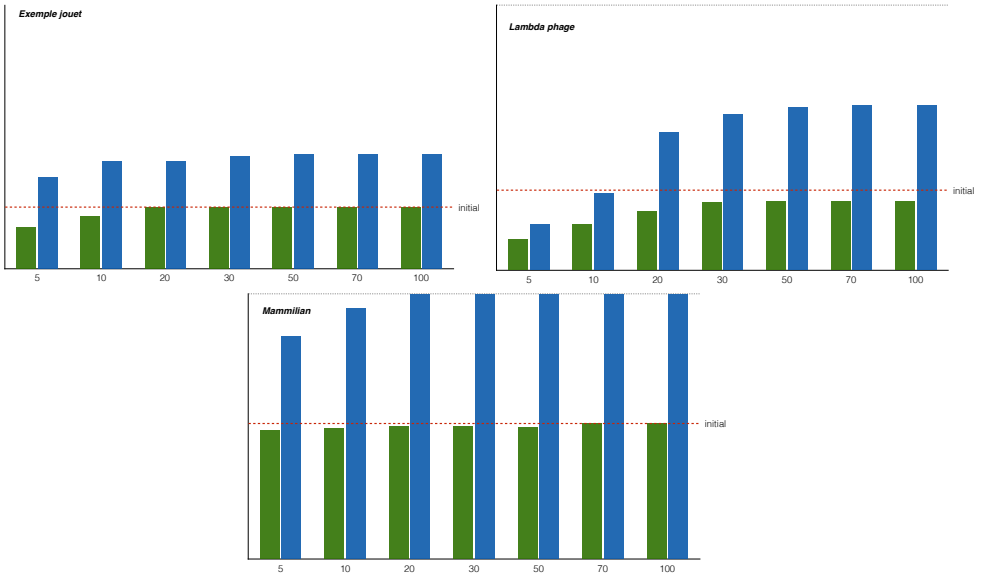


Figure 4.9 : Nombre des transitions locales temporisées apprises par l'algorithme 1, MoT-AN, à partir des chronogrammes issus des simulations des modèles du tableau 4.1 selon la taille de ces chronogrammes (valeurs en abscisses) : en vert les transitions locales temporisées qui existent aussi dans le modèle initial ($\mathcal{T}_{appris} \cap \mathcal{T}_{initial}$) et en bleu celles qui sont apprises en plus ($\mathcal{T}_{appris} \setminus \mathcal{T}_{initial}$). La ligne de référence en rouge indique le nombre des transitions locales temporisées du modèle initial.

temporisée n'est pas activée, alors elle ne sera pas apprise. En effet, une transition locale temporisée qui ne se manifeste pas en causant un changement de niveau d'un composant dans la dynamique du système, observée dans les chronogrammes, ne sera pas apprise par l'algorithme 1, MoT-AN. Par conséquent, étant donné que les simulations prises sont aléatoires, nous ne sommes pas sûrs que toutes les transitions locales temporisées soient activées, donc toutes les transitions du modèles ne vont pas être apprises. Ceci justifie le fait que pour le modèle "Lambda Phage" dans la figure 4.9, la barre verte n'atteint pas la ligne rouge qui indique le nombre des transitions locales temporisées du modèle initial.

D'autre part, on remarque dans la figure 4.9 ci-dessus, que le nombre des transitions locales temporisées apprises en plus (c'est-à-dire les $\tau \in \mathcal{T}_{appris} \setminus \mathcal{T}_{initial}$) augmente avec la taille des chronogrammes (les barres en bleu). En fait, elles sont cohérentes avec les chronogrammes pris en entrée et elles appartiennent à des modèles générés par MoT-AN mais elles n'existent pas dans le modèle initial. Ainsi, notre but dans la suite est de les éliminer. Donc, nous proposons quelques *filtres*, qui sont appliqués sur les modèles appris par MoT-AN afin de minimiser le plus possible le nombre des transitions locales temporisées qui sont apprises en surplus.

4.4 Raffinement du résultat de MoT-AN par des filtres

La méthode d'apprentissage MoT-AN, présentée dans la section 4.2 précédente, génère autant de T-AN que possible à partir des données de séries temporelles prises en entrée. Tous ces T-AN appris satisfont : d'une part, la dynamique du système étudié qui est illustrée par les données de séries temporelles et d'autre part, la dynamique des T-AN (introduite dans le chapitre 3). De plus ces modèles appris sont minimaux (c'est-à-dire pour chaque

changement dans les chronogrammes, il y a une seule transition qui l'explique) et ils sont cohérents (c'est-à-dire les transitions qui sont apprises tout en considérant qu'elles sont en conflit, elles appartiennent au même modèle).

Cependant, il peut y avoir des contradictions et/ou des incohérences dans ces modèles appris. Nous développons ces points dans cette section et nous présentons comment nous les traitons afin de les éliminer. En plus, il peut y avoir un grand nombre de modèles appris, donc, il serait difficile de distinguer le modèle correct qui modélise le système étudié.

Par conséquent, nous introduisons dans cette section, un ensemble de raffinements par des *filtres*. Ces filtres, permettent de ne conserver que les modèles appris qui sont considérés plus réalistes. Autrement dit, leurs comportements dynamiques sont plus proches de la dynamique du système réel.

4.4.1 Filtre de cohérence entre la dynamique du modèle et la dynamique des T-AN

Comme c'est décrit dans la section 3.3.2 en page 66, à propos de la sémantique de la dynamique des T-AN, nous imposons une activation en parallèle des transitions locales temporisées tant qu'il n'y a pas de conflits entre elles. Autrement dit, à un instant t, s'il y a au moins une transition locale temporisée τ qui n'est pas bloquée par une autre en cours et que τ est franchissable à t, alors τ devrait être activée à t et son résultat doit apparaitre à $t + \delta$ tel que $\mathsf{delai}(\tau) = \delta$.

Définition 4.4 (Filtre F1)**.** [Transitions synchrones et non en conflit] Soient $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$ un T-AN appris selon des chronogrammes Γ et selon le graphe d'influences χ entre les automates de Σ.
T-AN est dit cohérent avec la sémantique synchrone et non en conflit du chronogramme Γ si et seulement si $\forall \tau = a_i \xrightarrow[\delta]{\ell} a_j \in \mathcal{T}$, si τ est franchissable à un instant t dans Γ et $\nexists \tau' \in \mathcal{T}$, tel que τ' est en conflit avec τ (définition 3.3.2 en page 66), alors a doit changer d'état local de a_i vers a_j à $t + \delta$.

Le filtre correspondant vérifie si chaque modèle appris par l'algorithme 1, `MoT-AN`, est compatible avec la définition 4.4 ci-dessus des transitions synchrones et non en conflit. Si ce n'est pas le cas, c'est-à-dire s'il existe un modèle qui viole cette définition, alors il sera éliminé de la sortie de l'algorithme 1, `MoT-AN`. La violation est faite quand il existe une transition locale temporisée dans le T-AN appris, qui est franchissable à un instant t dans les chronogrammes Γ mais qui n'a pas été activée ; c'est-à-dire on n'observe pas dans les chronogrammes à l'instant $t + \mathsf{delai}(\tau)$ le changement d'états du composant qu'elle cible.

Exemple. Comme dans l'exemple jouet de la figure 4.8 en page 96, nous avons appris la transition locale temporisée $\tau = z_0 \xrightarrow[3]{a_0} z_1$ par le changement du composant z à l'instant $t = 3$ (dans $\mathcal{T}_{change(3)}$). Comparant avec l'ensemble des transitions locales temporisées du modèle initial, $\mathcal{T}_{initial}$ (figure 4.7 en page 96), on trouve que cette transition locale temporisée apprise n'y est pas. Donc, elle fait partie des transitions locales temporisées candidates apprises par `MoT-AN` mais qui ne sont pas correctes (en plus).

Ainsi, afin de voir si elle est à éliminer par ce filtre F1 (définition 4.4 ci-dessus), nous vérifions dans les chronogrammes Γ (figure 4.8 en page 96) si on trouve un instant où elle est franchissable et telle qu'elle n'est pas activée.

On trouve qu'en effet, à $t = 14$ $\tau = z_0 \xrightarrow[3]{a_0} z_1$ est franchissable car les conditions pour qu'elle soit jouable sont vérifiées : z est au niveau 0 et a est au niveau 0. En plus, puisque τ n'est pas en conflit avec aucune autre transition locale temporisée durant les 3 unités de temps de son activité ($\mathsf{delai}(\tau) = 3$) alors elle doit se terminer à $t = 17$. En revanche, z n'a pas changé de niveau à $t = 17$. Ainsi, dans les chronogrammes Γ, τ n'est pas activée à $t = 14$. On en déduit alors que τ n'est pas cohérente avec les chronogrammes Γ. Par conséquent, τ n'est pas cohérente avec la définition de la sémantique de la dynamique des T-AN. Donc, cette transition locale temporisée τ doit être éliminée par ce filtre F1. Par conséquent, dans ce cas, chaque modèle appris par MoT-AN contenant cette transition locale temporisée sera aussi exclu par ce filtre.

Dans la figure 4.10 ci-dessous, nous montrons les résultats obtenus après l'application du filtre F1. Nous pouvons remarquer la diminution du nombre des transitions locales temporisées apprises en plus (les barres en bleu) par rapport à celles de la figure 4.9 en page 110. En effet, nous avons pu éliminer plus de la moitié d'entre elles pour le modèle du Lambda-phage par exemple.

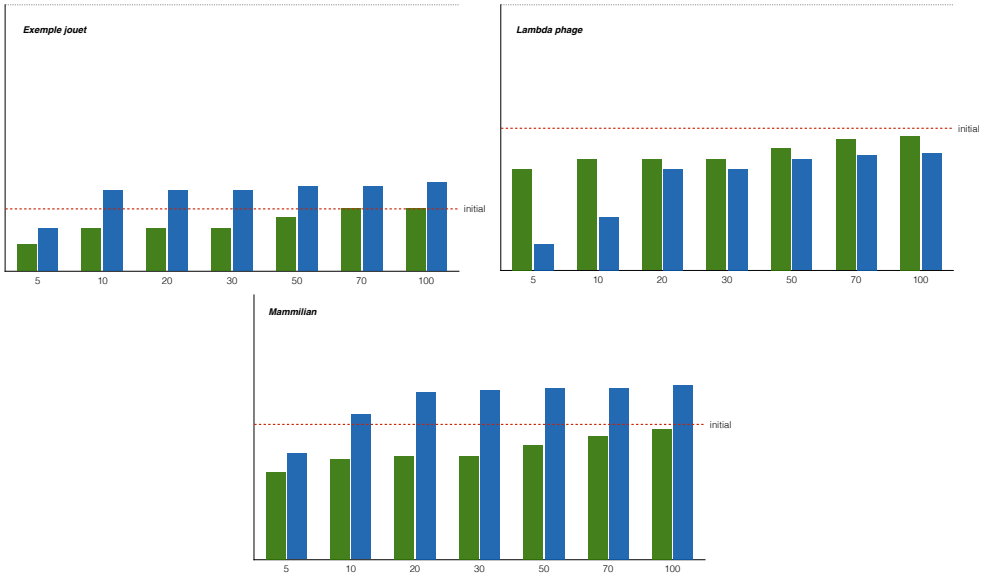


Figure 4.10 : Nombre des transitions locales temporisées apprises par l'algorithme 1, MoT-AN, après application du filtre F1 qui assure la non violation de la définition 4.4 ci-dessus. Ce filtre vérifie la cohérence entre la dynamique de chaque modèle appris et celle correspondantes aux chronogrammes. La taille de ces chronogrammes est écrite sur l'axe des abscisses. On trouve en vert les transitions locales temporisées qui sont correctes (i.e., existent aussi dans le modèle initial) et en bleu celles qui sont apprises en plus (i.e., n'existent pas dans le modèle initial). La ligne discontinue en rouge sert de référence pour le nombre des transitions locales temporisées du modèle initial.

4.4.2 Filtre basé sur la fréquence d'apparition des transitions locales temporisées dans les modèles appris

Tous les modèles générés par l'algorithme 1, MoT-AN, ont au moins une simulation possible qui reproduit exactement les chronogrammes donnés en entrée (voir théorème 4.1 en page 108). Il est donc intéressant de comparer tous ces modèles appris et de trouver les

transitions locales temporisées les plus partagées entre elles. En effet, si une transition locale temporisée est correcte, alors elle devrait avoir une forte probabilité d'apparaître dans un plus grand nombre de modèles par rapport à une autre qui n'est pas correcte. Ainsi, nous calculons dans la définition 4.5, la fréquence d'apparition $\mathsf{Freq}(\tau)$ d'une transition locale temporisée τ dans les modèles appris.

Définition 4.5 (Filtre F2)**.** [Fréquence d'apparence des transitions locales temporisées dans les T-AN appris] Soient $\mathcal{AN}^1 = (\Sigma, \mathcal{S}, \mathcal{T}^1)...\mathcal{AN}^m = (\Sigma, \mathcal{S}, \mathcal{T}^m)$ les modèles T-AN générés par l'algorithme 1, `MoT-AN`, avec $m \in \mathbb{N}$ le nombre total des T-AN appris. Soit $\mathsf{Freq}(\tau)$ la fréquence d'apparition d'une transition locale temporisée $\tau \in \mathcal{T}^i, 1 \leq i \leq m$ dans tous ces modèles appris :

$$\mathsf{Freq}(\tau) = \frac{\sum_{k=1}^{n} \mathsf{apparition_k}(\tau)}{m}$$

avec,

$$\mathsf{apparition_k}(\tau) := \begin{cases} 1 & \text{if } \tau \in \mathcal{T}^k \\ 0 & \text{if } \tau \notin \mathcal{T}^k. \end{cases}$$

Pour ce filtre F2, nous choisissons une valeur pour la fréquence minimale d'apparition de chaque transition locale temporisée : $\mathsf{minFreq} \in [0, 1]$. Ainsi, F2 ne maintient que les transitions locales temporisées τ dont la fréquence est supérieure ou égale à cette borne inférieure : $\mathsf{Freq}(\tau) \geq \mathsf{minFreq}$. Par conséquent, toute transition locale temporisée ayant une fréquence d'apparition inférieure à $\mathsf{minFreq}$ est éliminée par ce filtre F2. Par contre, il peut arriver que certaines transitions locales temporisées qui sont pourtant importantes soient aussi éliminées malencontreusement.

Revenons à notre exemple jouet, nous pouvons voir que la transition locale temporisée $b_1 \xrightarrow[3]{a_1} b_0$ est la seule qui est candidate pour le changement qui s'est produit à $t = 12$ (dans $\mathcal{T}_{change(12)}$). Ainsi, elle doit apparaitre dans tous les modèles T-AN appris et sa fréquence d'apparition est alors égale à 1.

Moins la transition apparaît dans les modèles appris, plus nous avons tendance à dire qu'elle n'est pas correcte (c'est-à-dire que ce n'est pas une transition locale temporisée qu'il faut apprendre). En effet, si un changement s'est produit plusieurs fois dans les chronogrammes, alors la transition locale temporisée qui est correcte (c'est-à-dire qui a effectivement causé ce changement) appartient à tous les ensembles des transitions locales temporisées candidates correspondants à ces changements. Par conséquent, quand l'algorithme 1, `MoT-AN`, génère des modèles T-AN, la transition locale temporisée qui est correcte apparaît dans plusieurs T-AN appris.

Exemple. Par exemple, dans la figure 4.8 en page 96 correspondant aux chronogrammes de l'exemple jouet, on remarque qu'à $t = 4$, $t = 9$ et $t = 17$, le même changement se produit : a change du niveau 0 au niveau 1. Selon le T-AN initial de la figure 4.7 en page 96, la transition locale temporisée qui est responsable de ce changement est $a_0 \xrightarrow[4]{\emptyset} a_1$. On note que dans tous les ensembles des transitions locales temporisées candidates apprises aux changements aux instants 4, 9 et 17 ; qui sont respectivement $\mathcal{T}_{change(4)}$, $\mathcal{T}_{change(9)}$ et $\mathcal{T}_{change(17)}$, il y a $a_0 \xrightarrow[4]{\emptyset} a_1$; (voir pages 95-99). Ainsi, puisqu'elle est candidate pour être responsable de plusieurs changements, quand `MoT-AN` génère tous

les modèle possibles, $a_0 \xrightarrow[4]{\emptyset} a_1$ apparaîtra dans plusieurs modèles. Ce qui fait augmenter sa fréquence d'apparition par rapport aux autres transitions locales temporisées candidates dans les ensembles $\mathcal{T}_{change(4)}$, $\mathcal{T}_{change(9)}$ et $\mathcal{T}_{change(17)}$.

Dans la figure 4.11 ci-dessous, nous donnons les résultats obtenus après l'application de ce filtre F2 qui ne garde que les transitions locales temporisées les plus fréquentes ; ayant une fréquence d'apparition dans les modèles appris par l'algorithme 1, MoT-AN qui est supérieure à une fréquence minimale choisie arbitrairement. Nous remarquons que le nombre des transitions locales temporisées apprises qui ne sont pas correctes (les barres en bleu) a énormément diminué par rapport à leur nombre pour le résultat sans les filtres, dans la figure 4.9 en page 110.

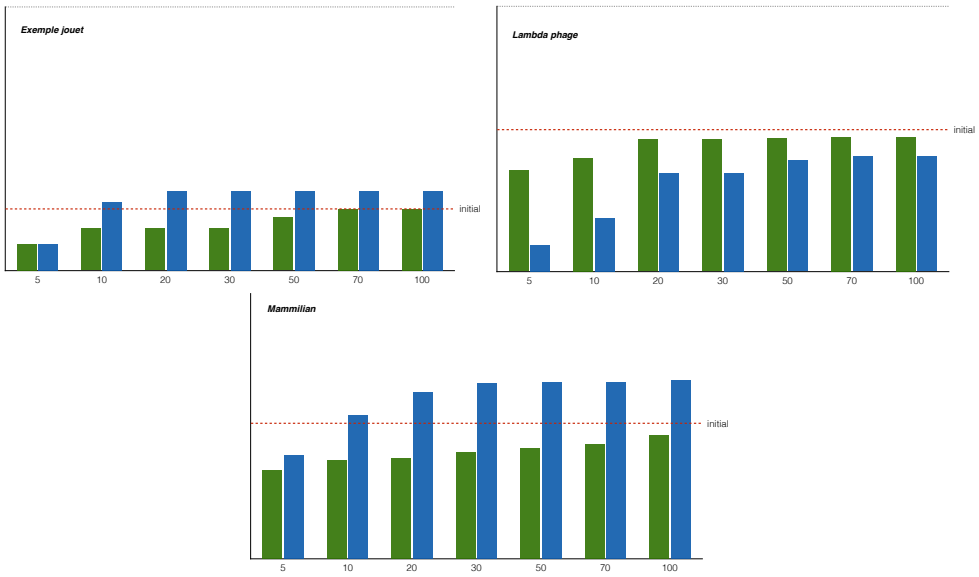


Figure 4.11 : Nombre des transitions locales temporisées apprises par l'algorithme 1, MoT-AN, après l'application du filtre F2 (définition 4.5 ci-dessus) qui ne garde que les transitions locales temporisées les plus fréquentes dans les modèles appris. Les barres en vert représentent les transitions locales temporisées qui sont correctes (i.e., existent dans le T-AN initial) et celles en bleu représentent celles apprises en plus (i.e., qui n'existent pas dans le T-AN initial). La ligne de référence en pointillés rouges, représente le nombre de transitions locales temporisées dans le T-AN initial.

4.4.3 Filtre associé au déterminisme des régulateurs entre les composants

Dans un modèle, nous ne voulons pas qu'un composant à un même niveau d'expression inhibe un composant dans certains cas et active ce même composant dans d'autres cas. En effet, cela ne semble pas avoir beaucoup de sens biologiquement. Dans un T-AN, un niveau d'expression discret d'un composant est l'état local de l'automate qui le représente. Dans la définition 4.6 ci-dessous, on formalise ceci ; dans un même T-AN appris, un composant b dans son état local b_k ne peut pas participer, d'une part, à une transition locale temporisée qui active un autre composant a, et d'autre part à une autre transition locale temporisée qui l'inhibe. Nous rappelons que dans la syntaxe des T-AN, si b_k participe à une transition locale temporisée τ, alors il appartient à sa condition $\mathsf{cond}(\tau)$.

Définition 4.6 (Filtre F3)**.** [Influence déterministe] Soient $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$ un T-AN, $\tau_1 \in \mathcal{T}$, $a, b \in \Sigma$, $a_i, a_j \in \mathcal{S}_a$ avec $i < j$ et $b_k \in \mathcal{S}_b$.
Si $\tau_1 = a_i \xrightarrow[\delta_1]{\ell_1} a_j$ telle que $b_k \in \ell_1$, alors $\forall \tau_2 \in \mathcal{T}$ telle que $\tau_2 = a_j \xrightarrow[\delta_2]{\ell_2} a_i$, $b_k \notin \ell_2$.

Par conséquent, ce filtre F3 garantit que dans un modèle T-AN appris, un automate, dans un état local précis, ne peut pas à la fois inhiber et activer un autre. Ainsi, ce filtre élimine tous les T-AN appris dans lesquels il existe des transitions locales temporisées qui violent la définition 4.6 ci-dessus, concernant les influences déterministes.

Exemple. Par exemple, dans les transitions locales temporisées apprises à partir des chronogrammes de l'exemple jouet de la figure 4.8 en page 96, les transitions locales temporisées $a_0 \xrightarrow[1]{z_1} a_1$ (dans $\mathcal{T}_{change(4)}$) et $a_1 \xrightarrow[1]{z_1} a_0$ (dans $\mathcal{T}_{change(5)}$) ne peuvent pas appartenir au même ensemble de transitions locales temporisées d'un T-AN appris. En effet, z_1 ne peut pas être à la fois responsable de l'activation et de l'inhibition de a.

Dans la figure 4.12 ci-dessous, nous donnons les résultats obtenus après l'application de ce filtre F3 sur les modèles appris par l'algorithme 1, `MoT-AN`. Nous remarquons que le nombre des transitions apprises qui ne sont pas correctes (en bleu) a énormément diminué par rapport à celui trouvé en exécutant la méthode `MoT-AN` sans les filtres (figure 4.9 en page 110).

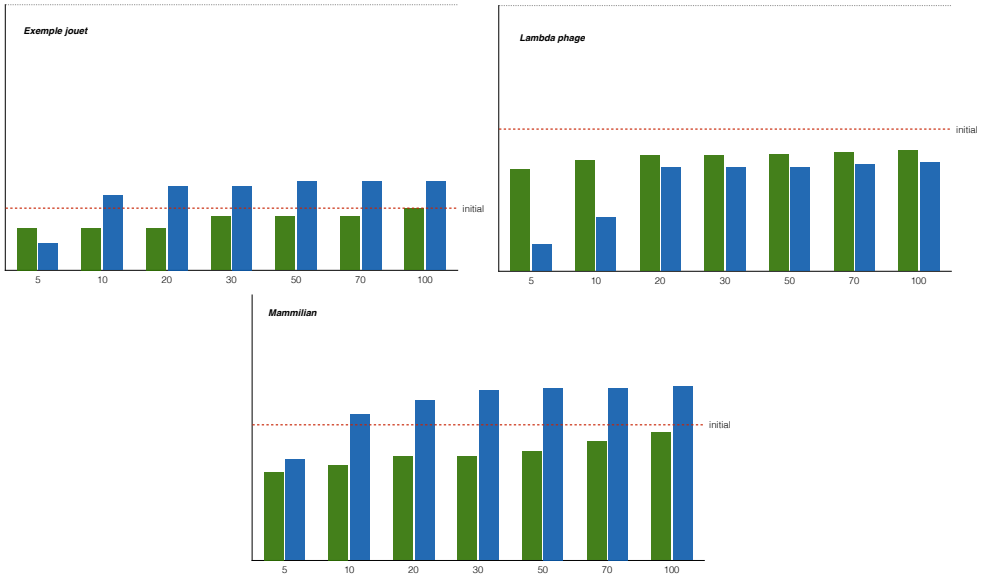


Figure 4.12 : Nombre des transitions locales temporisées apprises par l'algorithme 1, `MoT-AN`, après l'application du filtre F3 qui ne garde que les modèles T-AN dont l'ensemble des transitions locales temporisées est cohérent avec la définition 4.6 ci-dessus. Les barres en vert représentent les transitions locales temporisées qui sont correctes (i.e., existent dans le T-AN initial) et celles en bleu représentent celles apprises en plus (i.e., qui n'existent pas dans le T-AN initial). La ligne de référence en rouge discontinue représente le nombre de transitions locales temporisées dans le T-AN initial.

4.4.4 Filtre associé à un délai moyen d'une transition locale temporisée

Normalement dans un modèle T-AN, chaque transition locale temporisée n'a qu'un seul délai. Cependant, lorsque l'on apprend des modèles T-AN à partir des données de séries temporelles issues des expériences sur des systèmes biologiques réels (par exemple, des données provenant de lignées de cancers cellulaires ou des organes soumises au cycle de l'horloge circadienne...), on ne peut pas garantir que les données soient parfaites (i.e., non bruitées).

En fait, puisqu'il y a souvent du bruit dans les données, l'algorithme 1, MoT-AN, peut trouver des transitions locales temporisées qui sont presque identiques ; ne se différent que des valeurs de leurs délais. Ainsi, nous proposons de les fusionner en une seule ; ne conserver qu'une seule transition locale temporisée dont le délai est égal à un intervalle englobant tous les délais trouvés.

Définition 4.7 (Filtre F4)**.** [Intervalle des délais] Soit $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$ un T-AN, $\forall \tau_1, \tau_2, ..., \tau_n \in \mathcal{T}$ telles que $\tau_1 = a_i \xrightarrow[\delta_1]{\ell} a_j$, $\tau_2 = a_i \xrightarrow[\delta_2]{\ell} a_j$, ..., $\tau_n = a_i \xrightarrow[\delta_n]{\ell} a_j$ avec $a_i, a_j \in \mathcal{S}_a$, $\ell \in \wp(\mathbf{LS} \setminus \mathcal{S}_a)$ et $\delta_1 \neq \delta_2 \neq ... \neq \delta_n$ alors fusionner toutes les transitions locales temporisées $\tau_1, \tau_2, ..., \tau_n$ en une seule τ avec :

$$\tau = a_i \xrightarrow[[\delta_{min}, \delta_{max}]]{\ell} a_j$$

telle que,

$$\forall \tau_k = a_i \xrightarrow[\delta_k]{\ell} a_j, k \in \{1, ..., n\}, \delta_{min} \leq \delta_k \leq \delta_{max}.$$

Une autre alternative possible est de fusionner toutes les transitions locales temporisées qui sont identiques en une seule dont le délai est égal à une valeur moyenne de leurs délais (voir définition 4.8 ci-dessous). L'intuition est que, dans la pratique, s'il y a suffisamment des données de séries temporelles, les délais de ces transitions devrait tendre vers la valeur réelle.

Définition 4.8 (Filtre F4 révisé)**.** [Délai moyen] Soit $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$ un T-AN, $\forall \tau_1, \tau_2, ..., \tau_n \in \mathcal{T}$ telles que $\tau_1 = a_i \xrightarrow[\delta_1]{\ell} a_j$, $\tau_2 = a_i \xrightarrow[\delta_2]{\ell} a_j$, ..., $\tau_n = a_i \xrightarrow[\delta_n]{\ell} a_j$ avec $a_i, a_j \in \mathcal{S}_a$, $\ell \in \wp(\mathbf{LS} \setminus \mathcal{S}_a)$ et $\delta_1 \neq \delta_2 \neq ... \neq \delta_n$ alors fusionner toutes les transitions locales temporisées $\tau_1, \tau_2, ..., \tau_n$ en une seule τ avec :

$$\tau = a_i \xrightarrow[\delta_{avg}]{\ell} a_j$$

telle que :

$$\delta_{avg} = \frac{\sum_{k=1}^{n} \delta_k}{n}$$

Si nous voulons que chaque transition locale temporisée ait un seul délai dans un modèle T-AN appris (délai déterministe), dans la définition 4.9 ci-dessous, nous développons ainsi un filtre supplémentaire. Ce dernier garantit le fait que dans chaque T-AN généré, chaque transition locale temporisée n'a qu'un seul délai. Ainsi, s'il existe deux transitions locales

temporisées identiques mais qui ne se diffèrent que des valeurs de leurs délais, alors elles n'appartiennent jamais au même T-AN appris.

Définition 4.9 (Filtre F4 consolidé)**.** [Délai déterministe] Soient $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$ un T-AN, et $\tau_1 \in \mathcal{T}$ telle que $\tau_1 = a_i \xrightarrow[\delta_1]{\ell} a_j$, alors $\nexists \tau_2 \in \mathcal{T}$, telle que $\tau_2 = a_i \xrightarrow[\delta_2]{\ell} a_j$ avec $\delta_1 \neq \delta_2$.

Exemple. Par exemple, dans les transitions locales temporisées apprises à partir des chronogrammes de l'exemple jouet de la figure 4.8 en page 96, les transitions locales temporisées $a_0 \xrightarrow[1]{z_1} a_1$ (dans $\mathcal{T}_{change(4)}$) et $a_0 \xrightarrow[4]{z_1} a_1$ (dans $\mathcal{T}_{change(9)}$) ne peuvent pas appartenir au même ensemble de transitions locales temporisées d'un T-AN appris. En effet, la transition $a_0 \xrightarrow[\delta]{z_1} a_1$ ne peut pas avoir deux délais différents dans un même modèle T-AN appris.

Dans la figure 4.13 ci-dessous, nous donnons les résultats obtenus après l'application de ce filtre F4 consolidé sur les modèles appris par l'algorithme 1, `MoT-AN`. Nous remarquons que le nombre des transitions locales temporisées apprises qui ne sont pas correctes (les barres en bleu) a encore énormément diminué par rapport à leur nombre dans la figure 4.9 en page 110 sans les filtres.

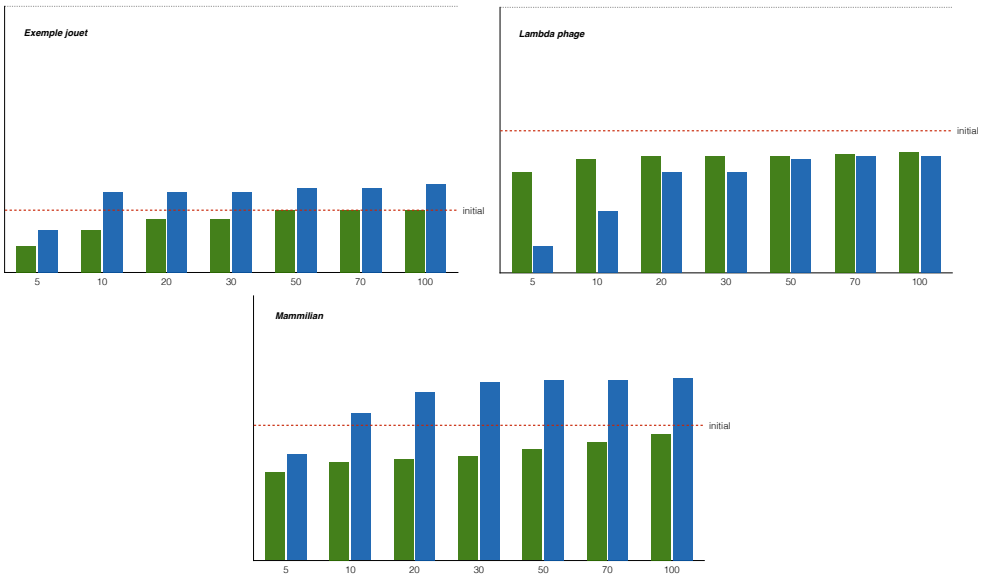


Figure 4.13 : Nombre des transitions locales temporisées apprises après application du filtre F4 consolidé qui assure le déterminisme des délais pour chaque transition locale temporisée dans chaque T-AN appris (définition 4.9).

Nous avons introduit dans cette section un ensemble de filtres qui sont utiles pour s'assurer au maximum de n'apprendre que les T-AN les plus cohérents avec les chronogrammes pris en entrée et ayant moins de contradiction. En effet, les filtres permettent d'éliminer un très grand nombre de modèles T-AN appris qui ne sont pas corrects (c'est-à-dire ne modélise pas le système réel). Ainsi, ils éliminent aussi les transitions locales temporisées apprises qui ne sont pas correctes. Nous notons que l'application de plusieurs filtres à

la fois est également possible. En fait, ceci permet d'optimiser davantage le résultat de l'apprentissage des modèles par `MoT-AN`. Il est à noter aussi que ces filtres peuvent être appliquer lors de la génération des modèles à l'étape 2 de l'algorithme 1 en page 93. Ceci est le cas des résultats des expériences que nous avons montrés dans cette section sur l'application des filtres.

Dans la section suivante nous présentons une méthode qui permet de réviser des T-AN existants à partir des données de série temporelles nouvellement fournies.

4.5 Révision des modèles

Dans cette section, nous présentons la méthode de la révision d'un T-AN existant (e.g., appris antérieurement) selon des données supplémentaires. Ces données peuvent être des nouvelles observations du système étudié illustrées par des nouvelles données de séries temporelles mais sous des nouvelles conditions (e.g., des perturbations par un ajout ou par une suppression d'un composant du système).

La méthode de la révision, consiste principalement à vérifier si le T-AN initialement proposé pour modéliser le système est aussi cohérent avec la dynamique donnée par les nouvelles observations expérimentales. Si ce n'était pas le cas, alors une *complétion* du T-AN initial serait nécessaire. En effet, au début il s'agit d'une complétion car nous ajoutons les transitions locales temporisées manquantes. Autrement dit, nous vérifions si tous les changements dans la dynamique du système peuvent être expliqués par des transitions locales temporisées existantes dans le T-AN initial, sinon nous apprenons des nouvelles transitions locales temporisées et nous les y ajoutons. D'autre part, une *suppression* des transitions locales temporisées du modèle initial pourrait-être faite par l'application d'un ou de plusieurs des filtres introduits dans la section 4.4 précédente.

Ainsi, le résultat final de la révision est un T-AN révisé selon les nouvelles données de séries temporelles. La figure 4.14 ci-dessous montre les entrées et la sortie de la méthode de la révision des T-AN `RevT-AN`.

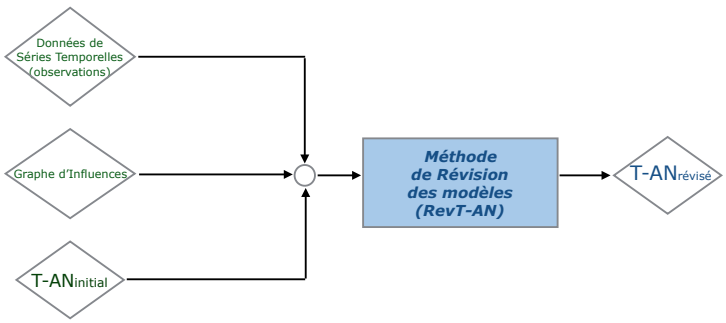


Figure 4.14 : La révision d'un T-AN initial modélisant un système donné selon des nouvelles données de séries temporelles.

La complétion des modèles a fait l'objet de nombreux travaux récents. Par exemple, dans (Akutsu et al., 2009), les auteurs ont ciblé la complétion des réseaux booléens stationnaires. Cette méthode a été affinée au fil des ans. Les travaux récents (Nakajima & Akutsu, 2013) se concentrent sur la complétion dans les réseaux génétiques variables dans le temps. Ce sont des réseaux dont la topologie ne change pas avec le temps, mais la nature des interactions (activation, inhibition ou absence d'interaction) entre les composants peut

changer à certains points temporels (temps fini). Ainsi, l'approche de complétion se réfère à la fois à l'addition et à la suppression des interactions, ce qui est en fait synonyme de révision. Elle a été appliquée avec succès à des études de cas biologiques ; par exemple le DREAM4 Challenge (Nakajima & Akutsu, 2014*b*) et dont la mise en œuvre a été améliorée par des heuristiques (Nakajima & Akutsu, 2014*a*). Cependant, leur méthode est limitée aux réseaux acycliques.

L'algorithme 2 ci-dessous décrit le pseudo-code de la méthode de la révision `RevT-AN`. Nous notons que `RevT-AN` est une extension de l'algorithme 1 de la méthode d'apprentissage, `MoT-AN`. En effet, une vérification supplémentaire est ajoutée après la génération de toutes les transitions locales temporisées candidates pour un changement donné. Cette vérification consiste à chercher dans le T-AN initial pris en entrée, s'il existe au moins une transition locale temporisée qui fait partie de l'ensemble des transitions locales temporisées candidates apprises au premier point de l'étape 1 de l'algorithme 2, `RevT-AN`, ci-dessous. Autrement dit, il faut vérifier s'il existe une transition locale temporisée qui explique chaque changement de niveaux d'un composant dans les chronogrammes. Sinon une complétion du modèle est effectuée par l'ajout d'une ou de plusieurs transitions locales temporisées. Ce processus est intégré dans le pseudo-code de l'algorithme 2, `RevT-AN` ci-dessous au deuxième point de l'étape 2.

Cette méthode de révision de modèles permet d'une part, de réviser les modèles existants et d'autre part de raffiner l'apprentissage dans le cas où nous avons plusieurs données de séries temporelles du même système mais sous des conditions différentes. En effet, après l'apprentissage d'un ensemble de modèles T-AN à partir des données de séries temporelles issues d'une expérience sous une première condition, cet ensemble de modèles appris sera révisé selon d'autres données issues d'autres expériences sous d'autres conditions.

Algorithm 2 RévT-AN : Révision des réseaux d'automates avec le temps.

Entrées :

- $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$: un T-AN à réviser ;
- $\Gamma = \bigcup_{a \in \Sigma} \Gamma_a$: chronogrammes des données de séries temporelles ;
- $\chi(N, E) = \bigcup_{a \in \Sigma} \chi_a(N_a, E_a)$: graphe d'influences entre les automates de $\mathcal{AN}$;
- $n \in \mathbb{N}^*$: le nombre maximal des régulateurs sur un composant dans une transition locale temporisée.

Sortie : Φ l'ensemble des T-AN qui réalisent les chronogrammes.

– Soit $\mathcal{T}_{revise} := \mathcal{T}$

– Étape 1 :
pour chaque point temporel t dans Γ où un composant a **change** son niveau discret de i vers j **faire** *//en T-AN c'est le changement de l'état local a_i vers a_j*

- **pour chaque** $\ell \in \wp(N_a), |\ell| \leq n$ **faire**
  //pour chaque combinaison possible des régulateurs de a

$$\tau := a_i \xrightarrow[\delta]{\ell} a_j$$

telle que,

si $\exists\ \tau' = c_k \xrightarrow[\delta_c]{\ell_c} c_l \in \varphi$ avec $a_i \in \ell_c$, $\Pi_t^c \geq \Pi_t^a$, $\forall b_x \in \ell$, $\Pi_t^c \geq \Pi_t^b$ **alors**
//τ est bloquée par τ' donc τ n'est activée que quand τ' se termine à Π_t^c

$$\delta = t - \Pi_t^c$$

sinon *//τ est activée à partir de l'instant qui est égal au maximum des //instants entre les derniers changements de a et des automates de ℓ*

$$\delta = t - max(\Pi_t^a, \Pi_t^d)$$

avec $d_x \in \ell \wedge \forall b_x \in \ell,\ b \neq d,\ \Pi_t^b \leq \Pi_t^d$.

ajouter τ dans $\mathcal{T}_{change(t)}$.
//$\mathcal{T}_{change(t)}$ est l'ensemble des transitions apprises par changement à l'instant t

- **si** $\mathcal{T}_{change(t) \subseteq \mathcal{T}}$ **alors** ne rien ajouter dans $\mathcal{T}_{revise}$;
  sinon, ajouter $C_{\mathcal{T}}(\mathcal{T}_{change(t)})$ dans $\mathcal{T}_{revise}$, à savoir le complémentaire de $\mathcal{T}_{change(t)}$ par rapport à $\mathcal{T}$

– Étape 2 :
Créer autant que possible dans Φ des T-AN appris, $\mathcal{AN}^k_{revise} = (\Sigma, \mathcal{S}, \mathcal{T}^k_{revise})$ avec $k \leq |\Phi|$ et $|\Phi|$ est le nombre total des T-AN appris. $\mathcal{T}^k_{revise} \subseteq \mathcal{T}_{revise}$ est l'ensemble **minimal** des transitions locales temporisées qui pourraient réaliser Γ, c'est-à-dire :

$$\forall \mathcal{AN}^k_{revise} = (\Sigma, \mathcal{S}, \mathcal{T}^k_{revise}) \in \Phi, \nexists \mathcal{T}^{k'}_{revise} \subset \mathcal{T}^k_{revise} \text{ tel que } \mathcal{T}^{k'}_{revise} \text{ peut réaliser } \Gamma,$$

autrement dit,

$\forall t$ un point temporel dans Γ, si $\exists a \in \Sigma$ tel que a change de niveau à t alors,

$$|\mathcal{T}^k_{revise} \cap \mathcal{T}_{change(t)}| = 1.$$

4.6 Application : système de l'horloge circadienne

4.6.1 Description et traitement des données de séries temporelles

Dans cette section, nous présentons les données qui ont été utilisées pour apprendre le modèle représentant le système de l'horloge circadienne. Ces données sont toutes issues d'expériences réalisées sur le foie de souris. L'objectif initial de ces expériences était d'analyser l'évolution des expressions des gènes du foie des souris à l'échelle du génome en fonction du temps, du génotype et des conditions environnementales (nourriture, température...). Suivant les expériences, les animaux ont donc été maintenus en conditions standard ou perturbées.

Les données sont des séries temporelles plus ou moins denses : une résolution de l'ordre de 1 à 4 heures suivant les expériences. Les données ont été obtenues avec la technologie : Microarrays Affymetrix. Elles ont été téléchargées à partir du site ArrayExpress à l'EBI[1]. On note que ces données sont déjà normalisées par l'EBI. La conversion des identifiants a été faite avec gprofiler[2].

Nous notons que ce travail est élaboré en coopération avec un biologiste de l'institut de Valrose en Biologie, Franck Delaunay, dans le cadre du projet ANR HyClock[3].

La figure 4.15 ci-dessous, montre les courbes qui décrivent les données de séries temporelles des gènes identifiés en tant que les plus importants pour l'étude du système de l'horloge circadienne : $Bmal1(C)$, $Bmal1(N)$, $Rev\text{-}Erb(N)$, $Per\text{-}Cry(N)$, $Cry1(C)$, $Cry2(C)$, $Per1(C)$, $Per2(C)$ et $Clock(N)$ tels que (N) et (C) indiquent que le composant se trouve respectivement dans le nucléotide ou dans le cytoplasme.

Pour réaliser la discrétisation des données des différentes variables retenues dans le modèle en évitant des profils complexes (0 ou 1 non contigus), les données de séries temporelles ont été ajustées avec une fonction sinusoïdale : méthode du cosinor (Refinetti, Cornélissen & Halberg, 2007), comme il est montré sur la figure 4.15 ci-dessous. On considère que pour qu'une oscillation autour de la valeur moyenne soit considérée comme significative et donc donne lieu à l'estimation de sa phase, il faut que la p-value pour l'amplitude d'oscillation soit inférieure à $0,005$.

[1] `http://www.ebi.ac.uk/arrayexpress/`

[2] `http://biit.cs.ut.ee/gprofiler/gconvert.cgi`

[3] `http://www.agence-nationale-recherche.fr/?Projet=ANR-14-CE09-0011.`

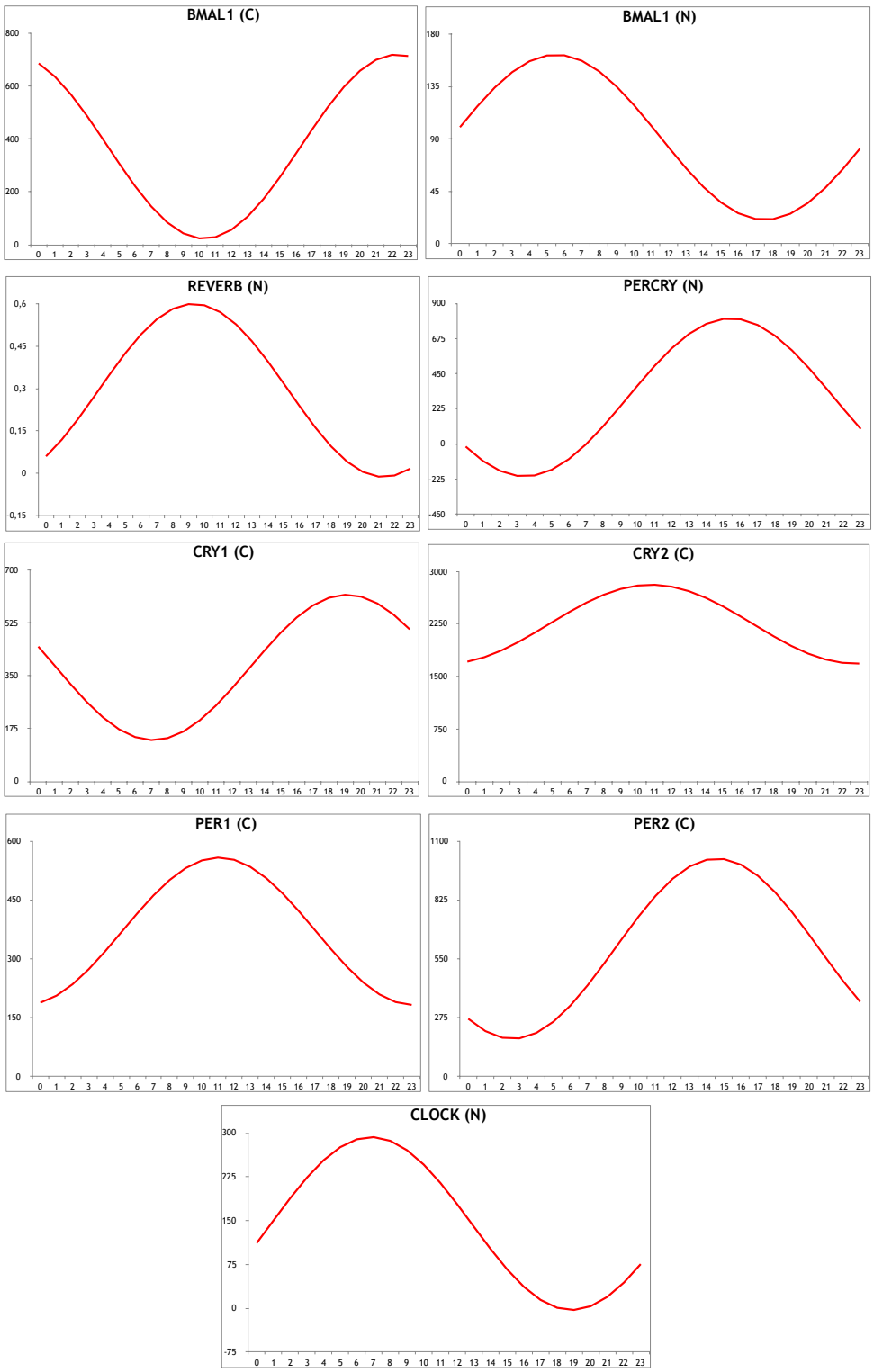


Figure 4.15 : Les données de séries temporelles des 9 gènes du système de l'horloge circadienne sur une journée de 24 heures : cycle lumineux durant les 12 premières heures et une obscurité constante sur les 12 heures suivantes, cycle qui est illustré par $Clock(N)$.

Ensuite, pour discrétiser on transforme les oscillations des données de séries temporelles (figure 4.15 ci-dessus) en des valeurs booléennes : 0 ou 1 pour les valeurs d'expression des gènes qui sont respectivement inférieures ou supérieures à la valeur du niveau moyen de chaque oscillation. Le résultat de la discrétisation est un ensemble de chronogrammes illustrés dans la figure 4.16 ci-dessous. Pour moins d'ambiguïté et pour la clarté des données, nous notons *Bmal1*(C), *Bmal1*(N), *Rev-Erb*(N), *Per-Cry*(N), *Cry1*(C), *Cry2*(C), *Per1*(C), *Per2*(C) et *Clock*(N) respectivement par *Bmal1-C*, *Bmal1-N*, *Rev-Erb*, *Per-Cry*, *Cry1*, *Cry2*, *Per1*, *Per2* et *Clock*.

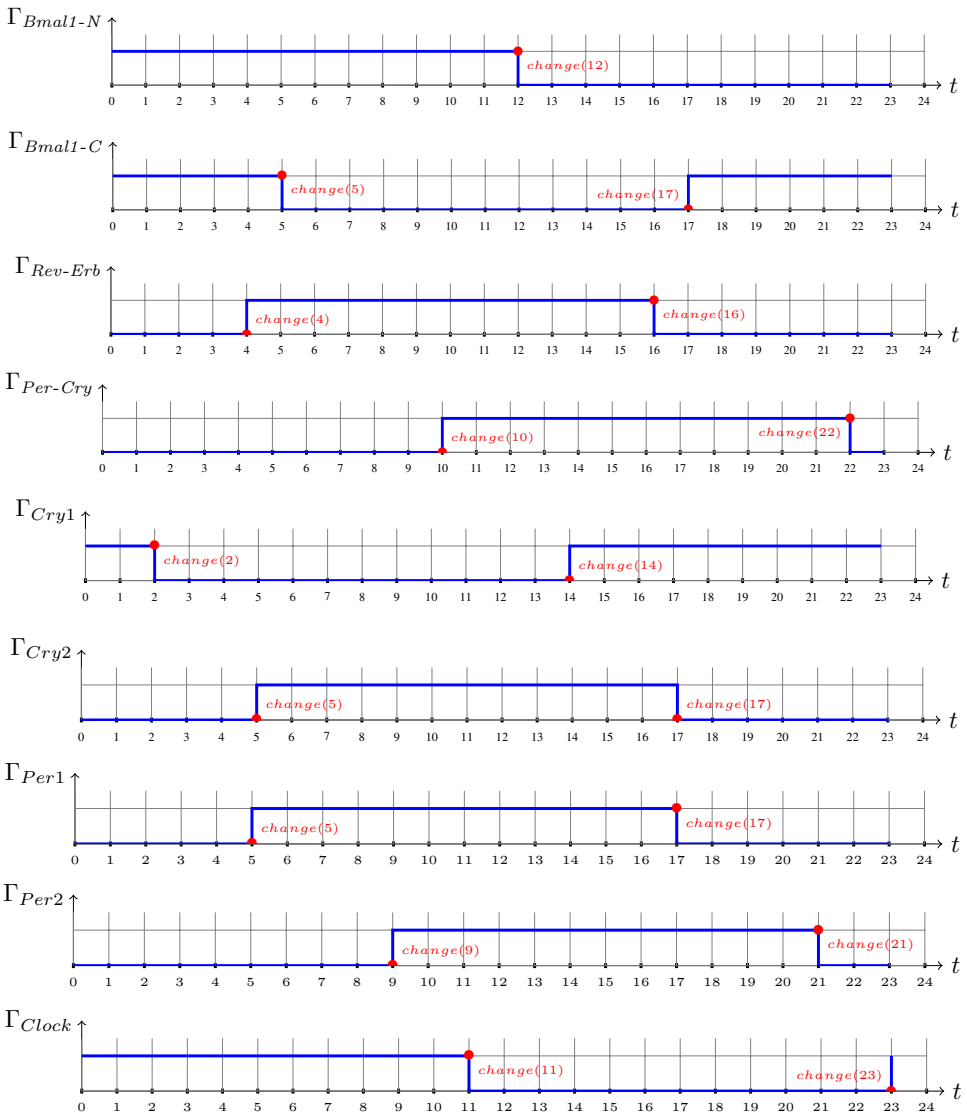


Figure 4.16 : Les chronogrammes trouvés après la discrétisation des données de séries temporelles des 9 composants du modèle du système de l'horloge circadienne de la figure 4.15 en page 122. Ils illustrent les évolutions discrètes des 9 composants sur 24 heures.

Pour apprendre le modèle représentant le système de l'horloge circadienne du foie, nous appliquons l'algorithme 1, `MoT-AN` (son pseudo-code est donné en page 93). Les chronogrammes pris en entrée sont ceux de la figure 4.15 ci-dessus. Et le graphe d'influences pris aussi en entrée est celui de la figure 4.17 ci-dessous.

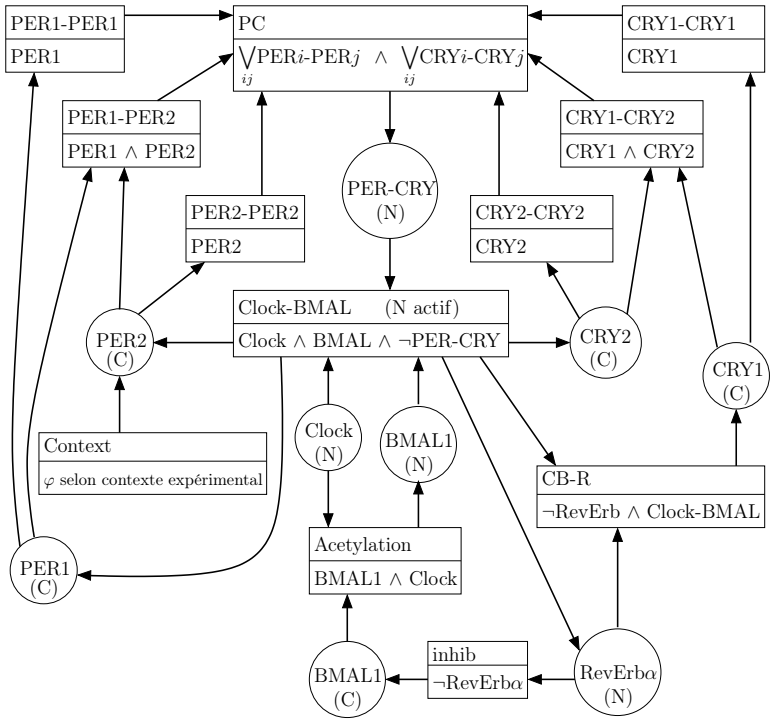


Figure 4.17 : Le graphe d'influences du modèle du système de l'horloge circadienne construit par Gilles Bernot, Jean-Paul Comet, Franck Delaunay, Morgan Magnin, Loïc Paulevé, Adrien Richard et Olivier Roux en 2012. Les cercles représentent les 9 composants. (N) et (C) indiquent que le composant se trouve respectivement dans le nucléotide ou dans le cytoplasme. Les rectangles représentent les multiplexes qui illustrent la coopération entre différents composants pour agir sur un autre composant du modèle.

Les multiplexes, dans le graphe d'influences du modèle de l'horloge circadienne de la figure 4.17, permettent de trouver directement les expressions logiques des coopérations entre les composants du modèle pour agir sur un autre composant. Nous rappelons que dans un T-AN, une coopération entre les composants est représentée dans la condition d'une transition locale temporisée. Par exemple, le multiplexe *Clock-Bmal* est actif dans le nucléotide quand *Clock* et *Bmal1-N* sont actifs (niveau discret est égal à 1) et que *Per-Cry* est inactif (niveau discret est égal à 0).

Ainsi, grâce aux multiplexes, on sait quelles sont les combinaisons correctes des régulateurs qui agissent sur chaque composant. Dans le graphe d'influences de la figure 4.17 ci-dessus, chaque composant est régulé par un seul multiplexe : *PC*, *Clock-Bmal*, *Acetylation*, *CB-R* ou *inhib*. Ceci facilite le processus d'apprentissage des transitions locales temporisées. Si τ est la transition locale temporisée recherchée responsable d'un changement quelconque d'un composant a, et $\ell = \mathsf{cond}(\tau)$, alors il n'est plus nécessaire de générer toutes les combinaisons possibles entre tous les régulateurs de a (au premier point de l'algorithme 1 en page 93) : $\ell \in \wp(N_a), |\ell| \leq n$ avec $n \in \mathbb{N}$ est le nombre maximal des régulateurs de a par transition. En effet, pour chaque changement d'état local d'un composant du système de l'horloge circadienne, ℓ est unique et est égale à l'unique multiplexe qui régule le composant changé (figure 4.17 ci-dessus).

Par exemple, pour trouver la transition locale temporisée responsable d'un changement de niveau de *Per-Cry*, il suffit d'étudier le chronogramme de *PC* et non pas tous les chronogrammes de tous ces régulateurs indépendamment (c'est-à-dire de *Per1*, *Per2*, *Cry1* et *Cry2*).

Ainsi, notre but est alors d'enrichir ce graphe d'influences en ajoutant à chaque influence d'un multiplexe sur un composant, des informations extraites des données de séries temporelles discrétisées (les chronogrammes de la figure 4.15 ci-dessus). Plus précisément, notre objectif est de trouver : (i) les signes des influences (activation ou inhibition), (ii) les états locaux des composants participant à chaque régulation, et (iii) les délais pour que cette régulation ait lieu. Ainsi, nous pouvons inférer le T-AN représentant le système de l'horloge circadienne.

Afin de faciliter l'apprentissage, nous avons calculé les chronogrammes des multiplexes. Ensuite, nous les avons aussi donnés en entrée à l'algorithme 1, `MoT-AN`. En fait, on peut les voir comme des portes logiques ; un multiplexe soit au niveau 1 ou au niveau 0 selon les niveaux de ces composants. On rappelle qu'un composant est actif s'il est au niveau 1. Et les symboles $\wedge$, $\vee$ et $\neg$ expriment respectivement un "et", un "ou" et un "non".

- $PC = \bigvee_{ij} Peri\text{-}Perj \wedge \bigvee_{ij} Cryi\text{-}Cryj$.
  $\Rightarrow$ *PC* est au niveau 1 si au moins l'un des $Peri$ est au niveau 1 et au moins l'un des $Cryi$ est au niveau 1.
- *Clock-Bmal*=*Clock* $\wedge$ *Bmal1-N* $\wedge \neg$*Per-Cry*.
  $\Rightarrow$ *Clock-Bmal* est au niveau 1 à un instant donné si et seulement si *Clock* et *Bmal1-N* sont tous les deux au niveau 1 et *Per-Cry* est au niveau 0.
- *Acetylation* = *Clock* $\wedge$ *Bmal1-C*.
  $\Rightarrow$ *Acetylation* est au niveau 1 si et seulement si *Clock* et *Bmal1-C* sont tous les deux au niveau 1.
- *inhib* = $\neg$*Rev-Erb*.
  $\Rightarrow$ *inhib* est au niveau 1 si et seulement si *Rev-Erb* est au niveau 0.
- *CB-R* = $\neg$*Rev-Erb* $\wedge$ *Clock-Bmal*.
  $\Rightarrow$ *CB-R* est au niveau 1 si et seulement si *Rev-Erb* est au niveau 0 et *Clock-Bmal* est au niveau 1.

En nous basant sur les expressions logiques de ces multiplexes et sur les chronogrammes des composants du système de l'horloge circadienne de la figure 4.16 en page 123, nous avons calculé les chronogrammes des multiplexes dans la figure 4.18 ci-dessous.

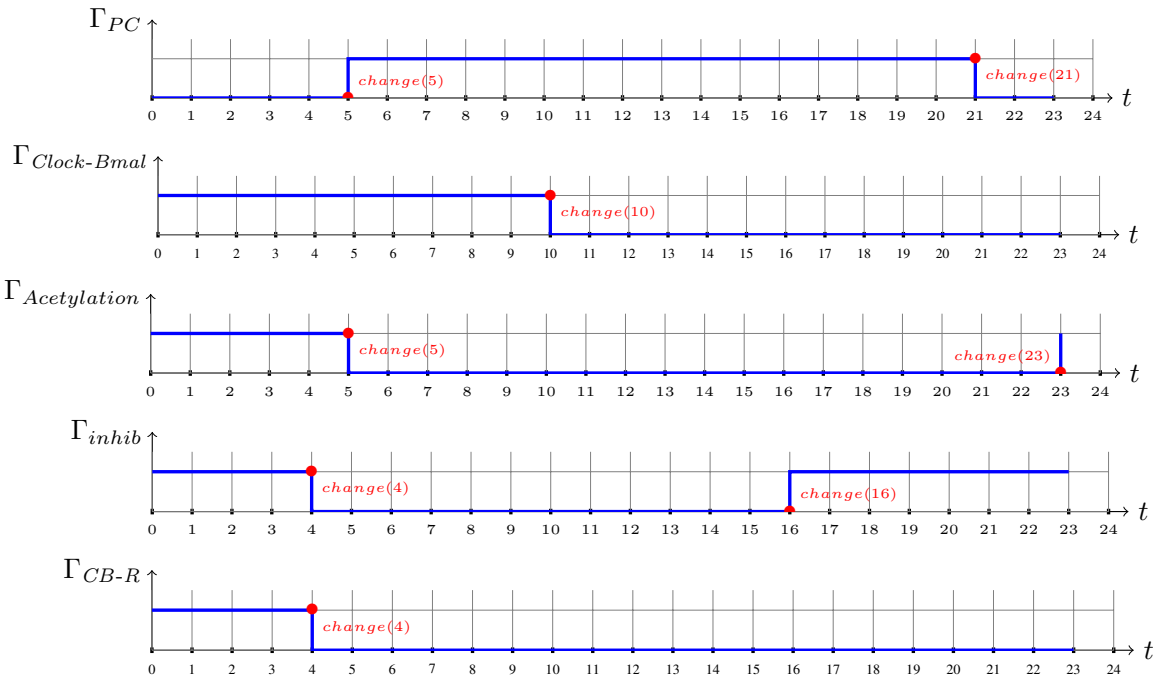


Figure 4.18 : Chronogrammes des multiplexes du modèle du système de l'horloge circadienne. Ils illustrent l'évolution discrète des multiplexes du modèle de la figure 4.17 en page 124 sur 24 heures.

Ainsi, nous avons tous les chronogrammes sur un cycle de 24 heures, illustrant l'évolution discrète de tous les composants biologiques (figure 4.16 en page 123) ainsi que des multiplexes (figure 4.18 ci-dessus) du modèle du système de l'horloge circadienne du foie.

4.6.2 T-AN appris par `MoT-AN` modélisant le système de l'horloge circadienne du foie

Le système de l'horloge circadienne oscille autour d'un cycle de 24 heures. Puisqu'on a les données de séries temporelles sur 24 heures (figure 4.15 en page 122), il suffit de répéter ces mêmes oscillations pour capturer la dynamique du système sur un cycle de 48 heures. Étant donné que les chronogrammes trouvés ci-dessous correspondent à des oscillations de chaque composant sur 24 heures (en valeurs discrètes), alors la répétition de ces chronogrammes permet de trouver leurs oscillations sur 48 heures.

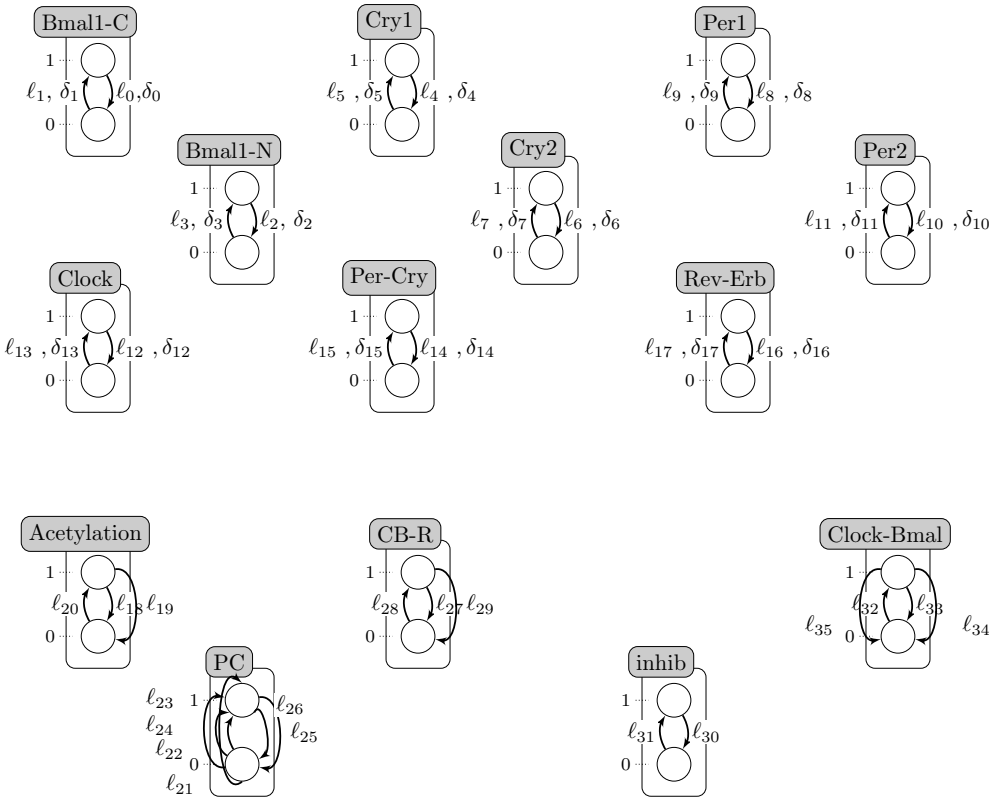
Ainsi, nous avons appliqué l'algorithme 1, `MoT-AN`, sur des chronogrammes de taille 48 afin d'avoir un meilleur résultat (c'est-à-dire pour que toutes les transitions locales temporisées soient apprises).

Le résultat de cet apprentissage est un seul T-AN qui représente le système de l'horloge circadienne du foie dont l'ensemble des transitions $\mathcal{T}_{appris}$ est détaillé ci-dessous.

$$
\begin{aligned}
\mathcal{T}_{appris} = \{ & \tau_1 = Per1_0 \xrightarrow[4]{\{Clock\text{-}Bmal_1\}} Per1_1, \\
& \tau_2 = Per1_1 \xrightarrow[7]{\{Clock\text{-}Bmal_0\}} Per1_0, \\
& \tau_3 = Per2_0 \xrightarrow[8]{\{Clock\text{-}Bmal_1\}} Per2_1, \\
& \tau_4 = Per2_1 \xrightarrow[11]{\{Clock\text{-}Bmal_0\}} Per2_0, \\
& \tau_5 = RevErb_0 \xrightarrow[3]{\{Clock\text{-}Bmal_1\}} RevErb_1, \\
& \tau_6 = RevErb_1 \xrightarrow[6]{\{Clock\text{-}Bmal_0\}} RevErb_0, \\
& \tau_7 = Cry1_1 \xrightarrow[1]{\{CB\text{-}R_1\}} Cry1_0, \\
& \tau_8 = Cry1_0 \xrightarrow[10]{\{CB\text{-}R_0\}} Cry1_1, \\
& \tau_9 = Cry2_0 \xrightarrow[4]{\{Clock\text{-}Bmal_1\}} Cry2_1, \\
& \tau_{10} = Cry2_1 \xrightarrow[7]{\{Clock\text{-}Bmal_0\}} Cry2_0, \\
& \tau_{11} = Per\text{-}Cry_0 \xrightarrow[5]{\{PC_1\}} Per\text{-}Cry_1, \\
& \tau_{12} = Per\text{-}Cry_1 \xrightarrow[1]{\{PC_0\}} Per\text{-}Cry_0, \\
& \tau_{13} = Bmal1\text{-}N_1 \xrightarrow[7]{\{Acetylation_0\}} Bmal1\text{-}N_0, \\
& \tau_{14} = Bmal1\text{-}N_0 \xrightarrow[1]{\{Acetylation_1\}} Bmal1\text{-}N_1, \\
& \tau_{15} = Bmal1\text{-}C_1 \xrightarrow[7]{\{inhib_0\}} Bmal1\text{-}C_0, \\
& \tau_{16} = Bmal1\text{-}C_0 \xrightarrow[1]{\{inhib_1\}} Bmal1\text{-}C_1 \\
& \}.
\end{aligned}
$$

Le T-AN appris contenant cet ensemble de transitions locales temporisées, $\mathcal{T}_{appris}$, est illustré sur la figure 4.19 ci-dessous. On y trouve aussi les automates qui représentent les 9 composants : *Bmal1-C*, *Bmal1-N*, *Cry1*, *Cry2*, *Per1*, *Per2*, *Clock*, *Per-Cry* et *Rev-Erb*.

D'autre part, on trouve dans la figure 4.19 ci-dessous, les automates qui représentent les multiplexes du graphe d'influences de la figure 4.17 en page 124 : *Acetylation*, *PC*, *CB-R*, *inhib* et *Clock-Bmal*. Puisque les automates des multiplexes représentent des coopérations entre les composants du modèle et qu'ils peuvent être vus comme des portes logiques, nous avons choisi que leurs transitions locales temporisées soient instantanées, c'est-à-dire sans délai. Comme il a été indiqué précédemment, leurs niveaux sont trouvés selon leurs formules logiques. En effet, l'état local des multiplexes est calculé à chaque instant t selon les états locaux des autres automates du réseau. Ils subissent une mise à jour directe dès que l'un de leurs composants change d'état local. Si par exemple, à un instant donné t, $Clock$ est activé (change d'état local de $Clock_0$ à $Clock_1$) alors à ce même instant t, le multiplexe $Acetylation = Clock \wedge Bmal1\text{-}C$ doit aussi s'activer (être à l'état local $Acetylation_1$). En fait, on peut dire que le changement d'états locaux des multiplexes est coordonné avec les changements des états locaux de leurs composants. Autrement dit, les changements sont faits en parallèle.



$\ell_0 = \{inhib_0\}, \delta_0 = 1$
$\ell_1 = \{inhib_1\}, \delta_1 = 1$
$\ell_2 = \{Acetylation_1\}, \delta_2 = 1$
$\ell_3 = \{Acetylation_0\}, \delta_3 = 7$
$\ell_4 = \{CB\text{-}R_1\}, \delta_4 = 2$
$\ell_5 = \{CB\text{-}R_0\}, \delta_5 = 10$
$\ell_6 = \{Clock\text{-}Bmal_0\}, \delta_6 = 7$
$\ell_7 = \{Clock\text{-}Bmal_1\}, \delta_7 = 5$
$\ell_8 = \{Clock\text{-}Bmal_0\}, \delta_8 = 7$
$\ell_9 = \{Clock\text{-}Bmal_1\}, \delta_9 = 5$
$\ell_{10} = \{Clock\text{-}Bmal_0\}, \delta_{10} = 11$
$\ell_{11} = \{Clock\text{-}Bmal_1\}, \delta_{11} = 9$
$\ell_{12} = \emptyset, \delta_{12} = 13$
$\ell_{13} = \emptyset, \delta_{13} = 11$
$\ell_{14} = \{PC_0\}, \delta_{14} = 1$
$\ell_{15} = \{PC_1\}, \delta_{15} = 7$
$\ell_{16} = \{Clock\text{-}Bmal_0\}, \delta_{16} = 2$
$\ell_{17} = \{Clock\text{-}Bmal_1\}, \delta_{17} = 10$
$\ell_{18} = \{Bmal1\text{-}C_0\}$
$\ell_{19} = \{Clock_0\}$
$\ell_{20} = \{Bmal1\text{-}C_1, Clock_1\}$
$\ell_{21} = \{Cry1_1, Per1_1\}$
$\ell_{22} = \{Cry1_1, Per2_1\}$
$\ell_{23} = \{Cry2_1, Per1_1\}$
$\ell_{24} = \{Cry2_1, Per2_1\}$
$\ell_{25} = \{Cry1_0, Cry2_0\}$
$\ell_{26} = \{Per1_0, Per2_0\}$
$\ell_{27} = \{Rev\text{-}Erb_1\}$
$\ell_{28} = \{Rev\text{-}Erb_0, Clock\text{-}Bmal_1\}$
$\ell_{29} = \{Clock\text{-}Bmal_0\}$
$\ell_{30} = \{Rev\text{-}Erb_1\}$
$\ell_{31} = \{Rev\text{-}Erb_0\}$
$\ell_{32} = \{Clock_1, Bmal1\text{-}N_1, Per\text{-}Cry_0\}$
$\ell_{33} = \{Clock_0\}$
$\ell_{34} = \{Bmal1\text{-}N_0\}$
$\ell_{35} = \{Per\text{-}Cry_1\}$

Figure 4.19 : Le T-AN modélisant le système de l'horloge circadienne du foie appris par la méthode `MoT-AN` à partir des données de séries temporelles de la figure 4.15 en page 122.

4.7 Discussion

Le sujet de ce chapitre porte sur l'apprentissage des modèles T-AN à partir des données de séries temporelles. Rappelons les étapes de notre approche qui peuvent se résumer ainsi :

- Pré-traiter les données de séries temporelles : trouver le graphe d'influences et les chronogrammes illustrant l'évolution discrète des composants du système à modéliser ;
- Identifier les instants auxquels les composants du système changent leurs niveaux d'expression discrets dans les chronogrammes ;
- Calculer les transitions locales temporisées candidates qui pourraient être responsables des changements de niveaux observés dans la dynamique du système représentée par

les chronogrammes ;
- Générer dans des T-AN appris les ensembles minimaux des transitions locales temporisées candidates qui sont cohérentes entre elles et peuvent réaliser la dynamique des chronogrammes ;
- Optimiser le résultat de l'apprentissage par l'application d'un ensemble de filtres.

L'objectif de la modélisation des systèmes est principalement d'avoir la possibilité d'analyser, de simuler et de comprendre leurs comportements. En effet, une fois que les biologistes ou les bioinformaticiens ont un modèle avec lequel il sont satisfaits, ils veulent souvent l'utiliser de manière prédictive, en réalisant des expériences virtuelles qui pourraient être difficiles, à réaliser avec le système réel en laboratoire. De telles expériences peuvent révéler des relations importantes et plutôt indirectes entre les composants du modèle qui seraient difficiles à prévoir autrement. C'est le sujet du chapitre suivant, qui porte sur l'analyse des propriétés dynamiques des systèmes modélisés avec le formalisme des réseaux d'automates.

Chapitre 5

Analyse de la dynamique des réseaux de régulation biologiques

La combinaison de nombreuses régulations simples entre les composants d'un réseau de régulation biologiques (RRB) mène souvent à des comportements complexes qui ne peuvent pas être compris intuitivement. Il est donc judicieux de développer des méthodes mathématiques appropriées pour identifier les propriétés dynamiques des RRB. Nous étudions ainsi, dans ce chapitre, principalement deux propriétés dynamiques dans les RRB modélisés avec le formalisme des réseaux d'automates. La première propriété dynamique est la vérification de l'*atteignabilité* d'un objectif (ensemble d'états locaux des automates) à partir d'un état global du réseau initialement donné. Une telle vérification permet par exemple de prédire une évolution dynamique du réseau. Ensuite, la deuxième propriété dynamique étudiée est l'identification des *attracteurs* dans les RRB. Un attracteur est un domaine piège minimal, c'est-à-dire une partie du graphe d'états de laquelle le réseau ne peut plus échapper. De telles structures présentent des composants terminaux de la dynamique et prennent la forme d'un état global stable (singleton) ou d'une composition complexe d'états globaux (non-singleton).

Ce travail a été publié dans (Ben Abdallah, Folschette, Roux & Magnin, 2015) et puis une version étendue et améliorée de ce travail est publiée dans (Ben Abdallah, Folschette, Roux & Magnin, 2017).

5.1 Introduction

Les analyses dynamiques des modèles permettent de vérifier la concordance entre un comportement observé du système et une propriété exprimée par le modèle qui le représente. En effet, en confrontant un modèle à des données expérimentales ou à des informations supplémentaires, l'analyse et la vérification formelle des propriétés dynamiques de ce modèle

apportent des preuves pour le valider ou pour le réfuter. En plus, elles permettent aussi de prédire certaines évolutions dynamiques du système modélisé. Par conséquent, les analyses dynamiques aident à la proposition des hypothèses concernant la dynamique du système modélisé afin d'en avoir une meilleure compréhension.

Dans ce chapitre, afin d'étudier l'analyse formelle des propriétés dynamiques des réseaux de régulation biologiques (RRB), nous utilisons le formalisme des réseaux d'automates (AN) qui a été proposé pour modéliser des systèmes concurrents ayant des composants avec quelques niveaux qualitatifs (Folschette et al., 2015). Nous avons précédemment introduit le formalisme des AN dans la section 3.2 en page 55 du chapitre 3. Et nous réintroduisons dans ce chapitre sa dynamique généralisée avec un peu plus de détails afin de pouvoir définir les méthodes qui vérifient ses propriétés dynamiques.

Notre objectif ici est de développer des méthodes particulières pour analyser d'une manière exhaustive les RRB modélisés avec le formalisme des AN. La première analyse que nous cherchons à effectuer consiste à vérifier l'*atteignabilité* d'un état local d'un automate (ou d'un ensemble d'états locaux de plusieurs automates) à partir d'un état global du AN initialement connu. Autrement dit, il s'agit de chercher à répondre à la question suivante : "partant d'un état global donné, est-il possible, en exécutant un nombre quelconque de transitions globales, d'atteindre un état global dans lequel tous les états locaux des automates choisis comme objectifs soient actifs ?".

D'autre part, le comportement à long terme de la dynamique des RRB est d'un intérêt particulier (Wuensche, 1998). En effet, à tout moment, un système peut tomber dans un *domaine piège*, qui fait partie de sa dynamique et duquel il ne peut pas s'échapper. Lors de son évolution, le système peut éventuellement tomber dans un nouveau et plus petit domaine piège (appelé *attracteur*) réduisant ainsi ses comportements futurs possibles (puisque les états précédents deviennent inaccessibles). Ce phénomène peut dépendre des perturbations biologiques ou d'autres phénomènes complexes. Une telle propriété dynamique est considérée comme une réponse caractéristique du système modélisé par le réseau. Par exemple, comme pour la différentiation en différents types de cellules dans l'organisme multicellulaire (Huang et al., 2005) ou la distinction entre les différentes tissus au cours du développement floral de l'*Arabidopsis thaliana* (Demongeot et al., 2010).

En outre, lors du raffinement d'un modèle représentant un système vivant, une des approches qui permettent de supprimer des incohérences ou encore de prédire des informations manquantes, consiste à comparer les attracteurs identifiés dans le modèle avec ceux observés au cours des expériences. Par exemple, le modèle du développement cellulaire de la *Drosophile melanogaster* est décrit par un réseau booléen ainsi que ses attracteurs (González et al., 2008).

Ainsi, la deuxième propriété dynamique étudiée dans ce chapitre consiste à identifier les attracteurs (i.e., domaines pièges minimaux) dans un AN. Dans le cadre des modèles qualitatifs comme les AN, un attracteur prend deux formes différentes. Une première forme est le *point fixe* (ou l'*état stable*) : un état global singleton à partir duquel le système n'évolue plus, appelé aussi un *point fixe*. Et la deuxième forme est l'*attracteur cyclique* : un ensemble (non singleton) d'états globaux dans lesquels le réseau cycle indéfiniment et n'en peut pas échapper.

Par conséquent, nous cherchons dans ce chapitre à énumérer exhaustivement tous les attracteurs d'un AN : les singletons ou les non singletons. En effet, la recherche des attracteurs cycliques de taille strictement supérieure à 1 n'est pas encore traité pour les

AN et non plus pour le PH (qui est une restriction des AN). Les travaux dans (Paulevé, Magnin & Roux, 2012) sur les RRB modélisés en PH, ont été uniquement focalisés sur l'identification des points fixes.

Nous introduisons dans la section 5.2 de ce chapitre la dynamique généralisée des AN. Ensuite, nous donnons les définitions formelles des propriétés dynamiques nécessaires pour la vérification de la propriété dynamique d'atteignabilité introduite dans la section 5.3 et pour la propriété dynamique qui porte sur l'identification des attracteurs introduite dans la section 5.4. Nous notons que ces définitions sont aussi utiles pour l'implémentation faite en Answer Set Programming (ASP) des programmes pour l'analyse des propriétés dynamiques des AN dans le chapitre 6.

5.2 Dynamique généralisée des réseaux d'automates

Nous présentons dans cette section la dynamique des RRB modélisés avec le formalisme des AN qui a été initialement introduite dans la section 3.2.1 du chapitre 3 en page 55. Nous notons que pour avoir une meilleure compréhension du contenu de ce chapitre, nous avons repris quelques définitions du chapitre 3. En effet, afin d'introduire les méthodes de l'analyse formelle de la dynamique d'un AN, des notations et des formulations sur sa dynamique générale sont nécessaires. Ainsi, dans la suite, nous présentons formellement, la sémantique de la dynamique des AN et nous définissons les propriétés dynamiques étudiées dans ce chapitre qui sont la vérification de l'atteignabilité et l'identification des attracteurs.

5.2.1 Définition des réseaux d'automates

Les interactions entre les composants d'un RRB sont modélisées d'une façon atomique et simple dans le formalisme des AN. En fait, un AN décrit, d'une manière atomique, les évolutions possibles d'un automate (représentant un composant unique) déclenchées par un ou plusieurs autres automates du réseau. Comparé à d'autres formalismes modélisant des RRB, la structure particulière du AN permet une analyse formelle des RRB avec des centaines de composants (Paulevé et al., 2012). De ce fait, les AN sont considérés comme étant bien adaptés pour l'analyse des propriétés dynamiques des RRB (Folschette et al., 2015).

La définition 5.1 ci-dessous introduit les AN comme un modèle avec des automates ayant un nombre fini de niveaux d'expression discrets appelés *états locaux*. Un *état local* d'un automate a est noté par a_i où a est l'identifiant de l'automate et i le niveau d'expression discret de a. A chaque instant, chaque automate a exactement un seul état local *actif*. L'ensemble des états locaux de tous les automates est appelé l'*état global* du réseau. Le changement des automates d'un état local actif vers un autre est assuré par les *transitions locales*. Leurs activités sont conditionnées par les états locaux actifs des autres automates du réseau (voir figure 5.1 ci-dessous).

Définition 5.1 (Réseaux d'Automates (AN))**.** Un *Réseau d'Automates* (Automata Network, AN) est un triplet $(\Sigma, \mathcal{S}, \mathcal{T})$ tel que :

- $\Sigma = \{a, b, \ldots\}$ est l'ensemble fini des identifiants des automates ;
- Pour tout $a \in \Sigma$, $\mathcal{S}_a = \{a_i, \ldots, a_j\}$ est l'ensemble fini des *états locaux* de l'automate a ; $\mathcal{S} = \prod_{a\in\Sigma} \mathcal{S}_a$ est l'ensemble fini des *états globaux* ; on note par $\mathbf{LS} = \cup_{a\in\Sigma}\mathcal{S}_a$ l'ensemble de tous les états locaux.
- Pour tout $a \in \Sigma$, $\mathcal{T}_a = \{a_i \xrightarrow{\ell} a_j \in \mathcal{S}_a \times \wp(\mathbf{LS} \setminus \mathcal{S}_a) \times \mathcal{S}_a \mid a_i \neq a_j\}$ est l'ensemble des *transitions locales* de l'automate a ; $\mathcal{T} = \bigcup_{a\in\Sigma} \mathcal{T}_a$ est l'ensemble de toutes les transitions locales dans le modèle.

Exemple. La figure 5.1 ci-dessous représente un exemple d'un AN.

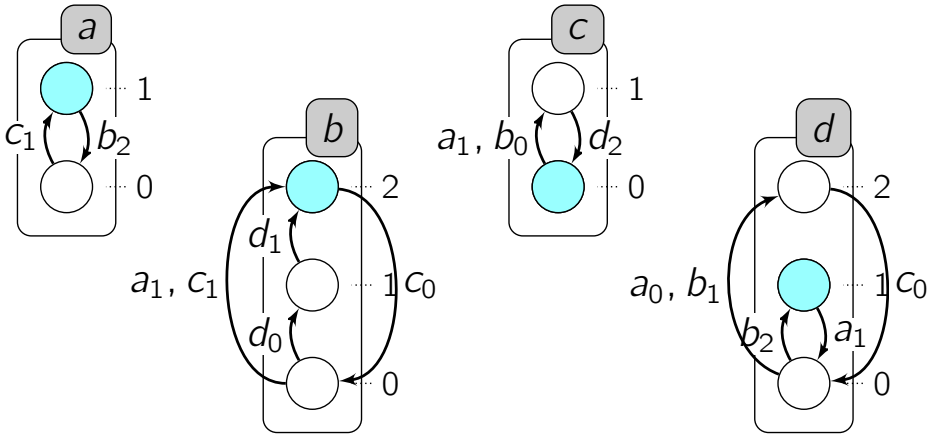


Figure 5.1 : Rappel de l'exemple de la figure 3.2 en page 56 d'un modèle AN ayant 4 automates a, b, c et d. Chaque boite représente un automate (modélisant par exemple un composant biologique), les cercles représentent leurs états locaux et les transitions locales sont représentées par des flèches étiquetées par les conditions de franchissement qui sont données par les états locaux d'autres automates. Les automates a et c sont au niveau discret 0 ou 1, et b et d ont 3 niveaux discrets 0, 1 et 2. Les états locaux colorés en bleu représentent l'état global du réseau $\langle a_1, b_2, c_0, d_1\rangle \sim 1201$.

Les interactions concurrentes entre les automates sont définies par un ensemble de transitions locales. Chaque transition locale τ est de la forme suivante : $\tau = a_i \xrightarrow{\ell} a_j$, où a_i et a_j sont des états locaux de l'automate a appelés respectivement *origine* et *destination* de τ. ℓ est la *condition* de τ et c'est l'ensemble des états locaux des autres automates différents de a (avec au plus un état local par automate). ℓ pourrait être égal à l'ensemble vide s'il s'agit d'une transition spontanée autonome.

Ainsi, pour la transition locale $\tau = a_i \xrightarrow{\ell} a_j$, on note : $\mathsf{ori}(\tau) = a_i$, $\mathsf{dest}(\tau) = a_j$ et $\mathsf{cond}(\tau) = \ell$.

5.2.2 La dynamique des réseaux d'automates

D'abord, nous rappelons ci-dessous les définitions introduites dans la sous-section 3.2.2 du chapitre 3 en page 57, des deux sémantiques les plus répandues dans la littérature et principalement étudiées dans ce chapitre : l'*asynchrone* (définition 5.3 en page 136) et le *synchrone* (définition 5.4 en page 136). Ces deux sémantiques diffèrent par le choix de

l'ensemble des transitions locales activées parmi celles qui sont *jouables* (définition 5.2 ci-dessous) permettant d'en déduire les transitions globales activées dans le graphe d'états correspondant.

Nous rappelons que $\zeta[a] = a_i$ désigne l'état local actif de a dans ζ qui est égal à a_i.

Ainsi, nous considérons qu'une transition locale $\tau = a_i \xrightarrow{\ell} a_j$ est jouable dans un état global ζ si et seulement si $\mathsf{ori}(\tau) = a_i$ est actif dans ζ (i.e., $\zeta[a] = a_i$) et que tous les états locaux dans $\mathsf{cond}(\tau) = \ell$ sont actifs dans ζ (i.e., $\forall b_k \in \ell$, $\zeta[b] = b_k$).

Définition 5.2 (Transitions locales jouables)**.** Soit $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$ un réseau d'automates et $\zeta \in \mathcal{S}$ un état global de $\mathcal{AN}$. L'ensemble des transitions locales jouables dans ζ noté par $\mathsf{J}(\zeta)$ est défini par :

$$\mathsf{J}(\zeta) = \{a_i \xrightarrow{\ell} a_j \in \mathcal{T} \mid \zeta[a] = a_i \wedge \forall b_k \in \ell,\ \zeta[b] = b_k\}.$$

Exemple. Soit $\zeta = \langle a_1, b_2, c_0, d_1\rangle \sim 1201$ un état global du AN de la figure 5.1 en page 134. $\mathsf{J}(1201)$ est l'ensemble des transitions locales jouables dans 1201 tel que :

$\mathsf{J}(1201) = \{a_1 \xrightarrow{\{b_2\}} a_0, b_2 \xrightarrow{\{c_0\}} b_0, d_1 \xrightarrow{\{a_1\}} d_0\}$.

$\mathsf{J}(1201)$ est calculé comme c'est indiqué dans la définition 5.2 et par exemple la transition locale $a_1 \xrightarrow{\{b_2\}} a_0$ est jouable dans 1201 car $\zeta[a] = a_1$, $\ell = \{b_2\}$ et $\zeta[b] = b_2$.

Dans cet état global $\zeta = 1201$, toute autre transition locale n'est pas jouable, car pour chacune, son origine ou bien au moins un des états locaux de sa condition n'est pas actif dans 1201. Par exemple, la transition locale $c_0 \xrightarrow{\{a_1, b_0\}} c_1$ n'est pas jouable car $\zeta[b] = b_2 \neq b_0$, alors que b_0 actif est une condition nécessaire pour que cette transition locale soit jouable.

L'activation successive des *transitions globales* entre les états globaux d'un AN correspond à un chemin qui s'intègre dans la dynamique du AN . Chaque transition globale est un ensemble de transitions locales. Ainsi, l'activation d'une transition globale implique l'activation de toutes les transitions locales qu'elle contient. Avant d'introduire la définition formelle des transitions globales jouables dans un état global, nous introduisons quelques sémantiques de la dynamique. En effet, calculer les transitions globales jouables pour un AN dans un état global donné dépend principalement de la sémantique de la dynamique.

Sémantiques de la dynamique des AN :

Différentes sémantiques de la dynamique peuvent être utilisées avec les formalismes des modèles discrets de type AN. Nous introduisons dans la suite les deux sémantiques les plus répandues dans la littérature : l'asynchrone et le synchrone. Il est à noter que le choix de la sémantique (asynchrone ou synchrone) amène à avoir différentes transitions globales activées. Ceci mène alors à avoir des évolutions dynamiques différentes. En effet, les transitions globales assurent l'évolution dynamique du réseau d'un état global vers un autre et qui est illustré par le graphe d'états.

Une transitions globale est jouable dans un état global donné et qui a été trouvée selon une sémantique asynchrone de la dynamique (définition 5.3 ci-dessous), contient exactement une seule transition locale parmi celles qui sont jouables dans l'état global considéré. Ainsi, chaque transition globale activée dans un chemin calculé selon une sémantique asynchrone ne change l'état local que d'un seul automate du réseau. Par conséquent, dans le graphe

d'états correspondant, deux états globaux successifs ont exactement une et une seule différence dans l'état local d'un automate. Cette différence est due à l'activation d'une seule transition locale contenue dans la transition globale activée dans le premier état global causant le changement de l'état local actif d'un automate dans l'état successeur.

Définition 5.3 (Sémantique asynchrone)**.** Soient $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$ un réseau d'automates, $\zeta \in \mathcal{S}$ un état global de $\mathcal{AN}$ et $\mathrm{J}(\zeta)$ l'ensemble des transitions locales jouables dans ζ. L'ensemble des *transitions globales jouables* dans ζ selon une sémantique asynchrone de la dynamique est :

$$U^{\text{asyn}}(\zeta) = \{\{a_i \xrightarrow{\ell} a_j\} \mid a_i \xrightarrow{\ell} a_j \in \mathrm{J}(\zeta)\}.$$

Exemple. Pour l'exemple de la figure 5.1 en page 134, on a l'ensemble des transitions locales jouables dans l'état global $\zeta = \langle a_1, b_2, c_0, d_1\rangle \sim 1201$ est : $\mathrm{J}(1201) = \{a_1 \xrightarrow{\{b_2\}} a_0, b_2 \xrightarrow{\{c_0\}} b_0, d_1 \xrightarrow{\{a_1\}} d_0\}$.

Ainsi, l'ensemble des transitions globales jouables dans 1201 selon la sémantique asynchrone est $U^{\text{asyn}}(1201)$ tel que :

$U^{\text{asyn}}(1201) = \{\{a_1 \xrightarrow{\{b_2\}} a_0\}, \{b_2 \xrightarrow{\{c_0\}} b_0\}, \{d_1 \xrightarrow{\{a_1\}} d_0\}\}$.

Par contre, le calcul d'une transition globale jouable selon la sémantique synchrone de la dynamique se fait différemment du calcul fait en asynchrone. En effet, elle présente l'ensemble de *toutes* les transitions locales jouables dans l'état global considéré. Sachant que ces transitions locales ne doivent pas être en concurrence pour qu'elles puissent s'activer en même temps. Des transitions locales sont dites en concurrence dans un état global ζ, si elles sont jouables dans ζ telles qu'elles appartiennent à un même automate a et elles le font évoluer vers des états locaux différents. Nous expliquons davantage cette notion de concurrences ainsi que ses conséquence sur la dynamique des AN dans la section 5.2.3 suivante.

Ainsi, nous notons qu'une transition globale trouvée selon la sémantique synchrone dans un état global ζ, $U^{\text{syn}}(\zeta)$, ne peut contenir qu'au plus une transition locale jouable dans chaque automate : $\forall u \in U^{\text{syn}}(\zeta)$ et $\forall a \in \Sigma$, $|u \cap \mathcal{T}_a| = 1$ (voir définition 5.4 ci-dessous).

Définition 5.4 (Sémantique synchrone)**.** Soient $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$ un réseau d'automates, $\zeta \in \mathcal{S}$ un état global et $\mathrm{J}(\zeta)$ l'ensemble des transitions locales jouables dans ζ. L'ensemble des transitions globales jouables dans ζ selon une sémantique synchrone de la dynamique est :

$$U^{\text{syn}}(\zeta) = \{u \subseteq \mathcal{T} \mid \forall a \in \Sigma, (\mathrm{J}(\zeta) \cap \mathcal{T}_a = \emptyset \Rightarrow u \cap \mathcal{T}_a = \emptyset) \wedge (\mathrm{J}(\zeta) \cap \mathcal{T}_a \neq \emptyset \Rightarrow |u \cap \mathcal{T}_a| = 1)\}.$$

Exemple. Pour le même exemple du modèle AN de la figure 5.1 en page 134, l'ensemble des transitions locales jouables dans l'état global $\zeta = \langle a_1, b_2, c_0, d_1\rangle \sim 1201$ est (le même que celui en asynchrone) : $\mathrm{J}(1201) = \{a_1 \xrightarrow{\{b_2\}} a_0, b_2 \xrightarrow{\{c_0\}} b_0, d_1 \xrightarrow{\{a_1\}} d_0\}$.

Ainsi, l'ensemble des transitions globales jouables dans 1201 selon la sémantique synchrone de la dynamique est $U^{\text{syn}}(1201)$, défini ci-dessous, contient toutes les transitions locales jouables dans 1201 :

$U^{\text{syn}}(1201) = \{\{a_1 \xrightarrow{\{b_2\}} a_0, b_2 \xrightarrow{\{c_0\}} b_0, d_1 \xrightarrow{\{a_1\}} d_0\}\}$.

Dans la suite, quand il n'y a pas d'ambiguïtés et quand les résultats sont applicables avec n'importe quelle sémantique (asynchrone, U^{asyn} ou synchrone, U^{syn}), nous notons

par U la sémantique choisie. La définition 5.5 ci-dessous définit formellement la transition globale dans un graphe d'états quelle que soit la sémantique U (qui peut être même une sémantique combinée entre l'asynchrone et le synchrone). Elle indique aussi que pour toutes les transitions locales τ qui appartiennent à une transition globale u, le changement d'états de $\mathsf{ori}(\tau)$ vers $\mathsf{dest}(\tau)$ doit apparaitre dans l'état global suivant.

Définition 5.5 (Transition globale)**.** Soient $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$ un réseau d'automates, $\zeta_1, \zeta_2 \in \mathcal{S}$ deux états globaux et U une sémantique de la dynamique (i.e., $U \in \{U^{\mathsf{asyn}}, U^{\mathsf{syn}}\}$). L'évolution de la dynamique du $\mathcal{AN}$ d'un état global ζ_1 vers un autre ζ_2 est assurée par la relation entre ces deux états globaux. Cette relation est appelée une *transition globale* trouvée selon une sémantique U, notée par $\zeta_1 \rightarrow_U \zeta_2$, et définie par :

$$\zeta_1 \rightarrow_U \zeta_2 \Longleftrightarrow \exists u \in U(\zeta_1) \text{ tel que } \zeta_2 = \zeta_1 \Cap \{\mathsf{dest}(\tau) \in \mathbf{LS} \mid \tau \in u\}.$$

ζ_2 est appelé le *successeur* de ζ_1, et ζ_1 est appelé le *prédécesseur* de ζ_2.

Rappelons que l'opérateur de recouvrement $\Cap$ est défini dans la définition 2.11 en page 44. Il signifie que pour tout état local $a_i \in \mathbf{LS}$, $\zeta \Cap a_i$ représente l'état global qui est identique à ζ, à l'exception de l'état local de a qui est remplacé par a_i : $(\zeta \Cap a_i)[a] = a_i \wedge \forall b \in \Sigma \setminus \{a\}, (\zeta \Cap a_i)[b] = \zeta[b]$. Si nous généralisons cette notation pour un ensemble d'états locaux $X \subseteq \mathbf{LS}$ contenant au maximum un état local pour chaque automate, alors on aurait $\forall a \in \Sigma, |X \cap \mathcal{S}_a| \leq 1$ où $|S|$ est le nombre des des éléments dans l'ensemble S ; dans ce cas, $\zeta \Cap X$ est l'état global ζ où l'état local de chaque automate a été remplacé par l'état local du meme automate dans X, s'il existe : $\forall a \in \Sigma, (X \cap \mathcal{S}_a = \{a_i\} \Rightarrow (\zeta \Cap X)[a] = a_i) \wedge (X \cap \mathcal{S}_a = \emptyset \Rightarrow (\zeta \Cap X)[a] = \zeta[a])$.

Exemple. Dans les deux exemples précédents, nous avons calculé pour le AN de la figure 5.1 en page 134, les ensembles des transitions globales jouables dans l'état global 1201. Dans le premier exemple en page 136, l'ensemble des transitions globales est calculé selon la sémantique asynchrone ($U^{\mathsf{asyn}}(1201)$) et puis dans le deuxième exemple, il est calculé selon la sémantique synchrone ($U^{\mathsf{asyn}}(1201)$). Les figures 5.4 et 5.5 en page 147 illustrent des parties des graphes d'états de transitions trouvés pour cet AN selon les sémantiques de la dynamique respectivement asynchrone et synchrone. Chaque transition globale est représentée par une flèche entre deux états globaux successifs.

Par exemple, on retrouve l'état global $\zeta = \langle a_1, b_2, c_0, d_1 \rangle \sim 1201$ (coloré en vert) dont les transitions globales jouables dans les sémantiques asynchrone et synchrones sont calculées dans les exemples précédents en pages 136. On retrouve qu'en asynchrone $\langle a_1, b_2, c_0, d_1 \rangle$ a 3 transitions globales jouables donc 3 successeurs potentiels ; $U^{\mathsf{asyn}}(1201) = \{\{a_1 \xrightarrow{\{b_2\}} a_0\}, \{b_2 \xrightarrow{\{c_0\}} b_0\}, \{d_1 \xrightarrow{\{a_1\}} d_0\}\}$. Alors qu'en synchrone, il n'a qu'une seule transition globale jouable : $U^{\mathsf{syn}}(1201) = \{\{a_1 \xrightarrow{\{b_2\}} a_0, b_2 \xrightarrow{\{c_0\}} b_0, d_1 \xrightarrow{\{a_1\}} d_0\}\}$. Cette unique transition globale jouable dans 1201 justifie le fait que 1201 n'a qu'un seul successeur.

Les chemins et leurs traces dans la dynamique des AN :

L'une des méthodes utilisées pour l'analyse de la dynamique d'un modèle discret, tel que le formalisme des AN, consiste à le faire évoluer à partir d'un état global initialement donné pour trouver une partie de son graphe d'états atteignable à partir de l'état global initial. En effet, chaque graphe d'états peut être vu comme un ensemble de séquences d'états globaux. Dans la dynamique des AN, nous appelons la séquence d'états globaux

atteignables successivement par un *chemin* noté par $H = (H_i)_{i\in[\![0;n]\!]} \in \mathcal{S}^{n+1}$ où chaque H_i est un état global.

Notons que, dans ce qui suit, lorsque nous définissons un chemin H de longueur n, nous utilisons la notation H_i pour désigner le i ème élément dans le chemin H, avec $i \in [\![0\,;n]\!]$. Nous utilisons aussi la notation $|H| = n$ pour désigner la longueur de H, permettant d'écrire : $H_{|H|}$ pour désigner le dernier élément de H (i.e., H_n).

La définition 5.6 ci-dessous introduit formellement la notion d'un chemin qui suit la dynamique d'un AN selon une sémantique donnée. Les chemins sont essentiels non seulement pour la vérification de l'atteignabilté d'un objectif à partir d'un état global initial donné mais aussi pour l'identification des attracteurs. En effet, pour atteindre un objectif à partir d'un état initial, il faut trouver un chemin qui vérifie cette propriété. Et pour identifier les états d'un attracteur, il faut trouver le chemin (ou les chemins) qui relit ces états. Nous donnons plus de détails dans la suite.

Définition 5.6 (Chemin)**.** Soient $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$ un réseau d'automates, U une sémantique donnée et $n \in \mathbb{N}$ un entier positif.
La séquence $H = (H_i)_{i\in[\![0;n]\!]} \in \mathcal{S}^{n+1}$ des états globaux est un *chemin de longueur n* si et seulement si : $\forall i \in [\![0\,;n-1]\!], H_i \rightarrow_U H_{i+1}$. Le chemin H est activé depuis l'état global initial H_0.

Exemple. Reprenons le modèle AN de la figure 5.1 en page 134. Soit la séquence H suivante présentant un chemin de longueur $n = 3$ activé depuis l'état global initial $H_0 = \langle a_1, b_2, c_0, d_1\rangle$ selon la sémantique asynchrone de la dynamique. Ce chemin peut être aussi visualisé dans la figure 5.4 en page 146 qui montre une partie du graphe d'états de ce modèle AN selon la sémantique asynchrone. L'état global initial de H est $H_0 = \langle a_1, b_2, c_0, d_1\rangle$ et il est coloré en vert dans la figure 5.4 en page 146 et l'état global final de H est $H_3 = \langle a_1, b_1, c_0, d_0\rangle$ et il est coloré en rouge.

$$H = (H_0 = \langle a_1, b_2, c_0, d_1\rangle\,; H_1 = \langle a_1, b_0, c_0, d_1\rangle\,; H_2 = \langle a_1, b_0, c_0, d_0\rangle\,; H_3 = \langle a_1, b_1, c_0, d_0\rangle)$$

Et plus simplement on peut écrire H ainsi :

$$H = (\langle a_1, b_2, c_0, d_1\rangle\,; \langle a_1, b_0, c_0, d_1\rangle\,; \langle a_1, b_0, c_0, d_0\rangle\,; \langle a_1, b_1, c_0, d_0\rangle)$$

Nous notons qu'en général un chemin de longueur n illustre n activations successives de transitions globales. Ainsi, il comporte donc jusqu'à $n+1$ états globaux. En l'occurrence pour l'exemple ci-dessus, H contient 4 états globaux car chaque état global n'est visité qu'une seule fois (sans répétition).

Par conséquent, un chemin $H = (H_i)_{i\in[\![0;n]\!]}$ n'est qu'une séquence d'états globaux compatible avec la dynamique du AN selon une sémantique donnée.

Chaque chemin H est caractérisé par sa *longueur n* qui correspond au nombre d'éléments dans sa séquence qui sont différents de son état global initial H_0. Autrement dit, on désigne par n le nombre de successeurs successifs de H_0 qui sont trouvés par les activations successives des transitions globales du AN. On peut dire aussi que n correspond au nombre de fois où il y a eu des activations des transitions globales au cours de H. Rappelons que l'activation d'une transition globale entraine le changement de l'état global du réseau vers un autre ($\forall i \in [\![0\,;n-1]\!], H_i \rightarrow_U H_{i+1}$). Et un chemin de longueur 0 est un singleton

($H = H_0$) c'est-à-dire qu'il n'y a aucune transition globale qui est activée et il s'agit donc d'un état global sans successeurs.

L'existence des cycles dans les AN peut induire des visites multiples des états globaux dans les chemins qui sont parcourus selon des cycles. En effet, si H contient un cycle alors il existe un état global qui apparait au moins 2 fois dans H ; par exemple si $\exists\ H_j = H_k$ avec $j, k \in [\![0\,;n-1]\!]$ et $j \neq k$ alors H parcourt un cycle entre H_j et H_k. Donc, un chemin de longueur $n \in \mathbb{N}$, n'implique pas qu'il couvre nécessairement $n+1$ états globaux différents.

Ainsi, afin de différencier un chemin qui est une séquence d'états globaux (et peut contenir des répétitions) de l'ensemble des états globaux qu'il parcourt, il est intéressant d'introduire la notion de *traces* de chemins. En fait, une trace d'un chemin est un ensemble (et non pas une séquence) des états globaux distincts visités tout au long du chemin (voir définition 5.7 ci-dessous).

Définition 5.7 (Trace)**.** Soient $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$ un réseau d'automates, U une sémantique de la dynamique donnée et $n \in \mathbb{N}$ un entier positif. La séquence $H = (H_i)_{i\in[\![0;n]\!]} \in \mathcal{S}^{n+1}$ des états globaux est un chemin de longueur n. La *trace* correspondante à un tel chemin H est l'ensemble des états globaux qui ont été visités :

$$\mathsf{trace}(H) = \{H_j \in \mathcal{S} \mid j \in [\![0\,;n]\!]\}.$$

Exemple. Reprenons le modèle AN de la figure 5.1 en page 134. Soit le chemin H suivant de longueur $n = 6$ activé depuis l'état global initial $H_0 = \langle a_1, b_2, c_1, d_1\rangle$ selon la sémantique asynchrone de la dynamique :

$$H = (H_0 = \langle a_1, b_2, c_1, d_1\rangle\,; H_1 = \langle a_0, b_2, c_1, d_1\rangle\,; H_2 = \langle a_1, b_2, c_1, d_1\rangle\,; H_3 = \langle a_1, b_2, c_1, d_0\rangle\,;$$
$$H_4 = \langle a_0, b_2, c_1, d_0\rangle\,; H_5 = \langle a_0, b_2, c_1, d_1\rangle\,; H_6 = \langle a_1, b_2, c_1, d_1\rangle).$$

Ainsi, $\mathsf{trace}(H) = \{\langle a_1, b_2, c_1, d_1\rangle, \langle a_0, b_2, c_1, d_1\rangle, \langle a_1, b_2, c_1, d_0\rangle, \langle a_0, b_2, c_1, d_0\rangle\}$.

On a alors, $|\mathsf{trace}(H)| = 4$. Ce chemin H peut être visualisé dans la figure 5.4 en page 146 dont les 4 états globaux de sa trace sont colorés en jaune.

S'il y a des visites multiples des états globaux au cours d'un chemin H (c'est-à-dire des répétitions), alors la taille de sa trace sera nécessairement strictement inférieure au nombre maximal de ses éléments. Ce dernier est égal à $n+1$ pour un chemin de longueur n. Autrement dit, s'il y a des cycles dans un chemin H qui est de longueur n, alors $|\mathsf{trace}(H)| < n+1$. En occurrence, pour l'exemple précédent, on a $|\mathsf{trace}(H)| = 4 < n+1 = 7$. En effet, il y a des cycles dans H qui sont déduits par les répétitions de certains états globaux au cours de H : $H_0 = H_2 = H_6$ et $H_1 = H_5$.

D'une façon générale, on peut trouver la taille de la trace d'un chemin H en soustrayant le nombre de fois où il y a eu des répétitions des états globaux au cours de H. En effet, plus il y a des états globaux répétés dans un chemin, moins il y a d'états globaux dans sa trace. Ainsi, nous formalisons dans le lemme 5.1 ci-dessous la formule qui calcule la taille de la trace d'un chemin quelconque à partir de sa longueur et son nombre correspondant de répétitions des états globaux.

Par conséquent, il faut calculer tout d'abord, le nombre de répétitions des états globaux dans le chemin. Et la définition 5.8 ci-dessous, introduit l'ensemble $\mathsf{sr}(H)$ qui contient les

indices des éléments de H auxquels il y a des répétitions des états globaux. Autrement dit, c'est le nombre de fois où les états globaux sont apparus au moins pour la deuxième fois dans H.

Définition 5.8 (Répétitions dans un chemin)**.** Soient $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$ un réseau d'automates, $n \in \mathbb{N}$ un entier positif et $H = (H_i)_{i \in [\![0;n]\!]} \in \mathcal{S}^{n+1}$ un chemin de longueur n. L'ensemble des *répétitions* dans H est donné par :

$$\mathsf{sr}(H) = \{i \in [\![1\,;n]\!] \mid \exists j \in [\![0\,;i-1]\!], H_j = H_i\}.$$

$\mathsf{sr}(H)$ qui est l'ensemble de répétitions dans un chemin H compte les indices d'états globaux qui existent ailleurs dans H avec un indice inférieur. Ainsi, la taille de $\mathsf{sr}(H)$ (i.e., $|\mathsf{sr}(H)|$) représente le nombre total des répétitions dans H. Par conséquent, ayant la longueur de H et les indices des états globaux qui y sont répétés, il est facile de calculer le nombre exact des états globaux distincts visités tout au long de ce chemin. Autrement dit, il s'agit de trouver la taille de sa trace.

Lemme 5.1 (Taille de la trace d'un chemin)**.** *Soit H un chemin de longueur n. Le nombre des éléments (i.e., des états globaux) de sa trace est donné par :*

$$|\mathsf{trace}(H)| = n + 1 - |\mathsf{sr}(H)|.$$

Démonstration. Soit H un chemin de longueur n, donc $\mathsf{trace}(H)$ contient au plus $n+1$ états globaux. Soit $|\mathsf{sr}(H)|$ le nombre de répétitions dans H. Alors le nombre des états globaux distincts visités au cours de H est égal au résultat de la soustraction du nombre des indices des états globaux répétés dans H (i.e., $|\mathsf{sr}(H)|$) du nombre total de tous les indices des états globaux visités au cours de H (i.e., $n+1$) : $|\mathsf{trace}(H)| = n + 1 - |\mathsf{sr}(H)|$. □

Nous notons que quand il n'y a aucune répétition dans un chemin H de longueur n alors $\mathsf{sr}(H) = \emptyset$ et donc $|\mathsf{sr}(H)| = 0$. On a donc le nombre total des états globaux visités au cours de H est exactement égal à : $|\mathsf{trace}(H)| = n + 1$.

Exemple. On peut vérifier le lemme 5.1 ci-dessus par le chemin H de l'exemple précédent en page 139 qui est de longueur $n = 6$.

On trouve que, l'état global $\langle a_1, b_2, c_1, d_1\rangle$ est visité 3 fois : à H_0, H_2 et H_6. Ainsi, selon la définition 5.8 ci-dessus, cet état global est répété 2 fois dans H : à H_2 et H_6 (i.e., aux éléments d'indices 2 et 6). En effet, la première apparition d'un état global dans un chemin H n'est pas comptée dans l'ensemble de répétions $\mathsf{sr}(H)$ (voir définition 5.8 ci-dessus).

D'autre part, l'état global $\langle a_0, b_2, c_1, d_1\rangle$ est visité 2 fois dans H : à H_1 et H_5. Ainsi, il est répété 1 seule fois à H_5 (i.e., dans l'élément d'indice 5).

Par conséquent, on a $\mathsf{sr}(H) = \{2, 6, 5\}$ et $|\mathsf{sr}(H)| = 3$. Sachant que la longueur de H est égale à $n = 6$, alors $|\mathsf{trace}(H)| = 6 + 1 - 2 - 1 = 4$.

Ce résultat est conforme à celui trouvé dans l'exemple précédent en page 139 dans lequel nous avons calculé les 4 états globaux de $\mathsf{trace}(H)$.

5.2.3 Déterminisme et non-déterminisme

D'une façon générale, quelle que soit la sémantique de mise à jour que la dynamique du réseau suit, il existe des états globaux qui ont plusieurs successeurs. Ce choix multiple de successeurs à partir d'un état global crée un *non-déterminisme* dans la dynamique.

Dans le cas de **la sémantique asynchrone**, le non-déterminisme provient principalement du fait qu'une et une seule transition locale est franchie parmi toutes celles qui sont jouables (voir définition 5.3 en page 136). En effet, toutes les transitions locales jouables, une fois activées, font évoluer le réseau vers des états globaux différents. Ainsi, pour un état global donné $\zeta \in \mathcal{S}$, dès que $|\mathrm{J}(\zeta)| > 1$, plusieurs successeurs peuvent exister. Rappelons que $\mathrm{J}(\zeta)$ est l'ensemble des transitions locales jouables dans l'état global ζ (définition 5.2 en page 135).

Dans la figure 5.4 en page 146, le graphe d'états illustre un exemple d'une évolution non déterministe : à partir d'un état global, il existe plusieurs flèches sortantes vers ses successeurs. En l'occurrence, l'état global $\langle a_1, b_2, c_0, d_1\rangle$ (coloré en vert) a trois successeurs : $\langle a_0, b_2, c_0, d_1\rangle$, $\langle a_1, b_0, c_0, d_1\rangle$ et $\langle a_1, b_2, c_0, d_0\rangle$.

En ce qui concerne **la sémantique synchrone** (voir définition 5.4 en page 136), le non-déterminisme dans l'évolution dynamique d'un réseau ne se produit que quand il y a des transitions locales qui sont concurrentes (voir définition 5.9 ci-dessous). En fait, deux transitions locales sont dites concurrentes dans un état global ζ, si elles agissent sur un même automate a, telles qu'elles sont toutes les deux jouables dans ζ, et qu'elles font évoluer a vers deux états locaux distincts. Cette concurrence apparaît seulement quand les deux transitions locales ont des destinations différentes mais la même origine et des conditions compatibles dans un même état global ; c'est-à-dire tous les états locaux des automates dans leurs conditions peuvent être actifs dans un même état global.

Définition 5.9 (Transitions locales concurrentes)**.** Soient $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$ un réseau d'automates, $a \in \Sigma$ est un automate et $\tau_1, \tau_2 \in \mathcal{T}_a$ deux transitions locales appartenant à l'automate a. τ_1 et τ_2 sont *concurrentes* si et seulement si :

$$\exists \zeta \in \mathcal{S} \text{ tel que } \tau_1 \in \mathrm{J}(\zeta) \wedge \tau_2 \in \mathrm{J}(\zeta) \wedge \mathsf{dest}(\tau_1) \neq \mathsf{dest}(\tau_2).$$

Les transitions locales concurrentes présentent la seule cause du non-déterminisme qui peut être observée dans le graphe d'états d'un AN calculé selon la sémantique synchrone. En effet, selon cette sémantique toutes les transitions locales qui sont jouables dans un état global doivent être activées dans la même transition globale. Ceci est à l'exception de celles qui ne peuvent pas s'activer en parallèle (i.e., celles qui sont concurrentes). Effectivement, les transitions locales concurrentes une fois activées, vont changer l'état local actif du même automate vers deux états locaux actifs différents. Et puisque chaque automate ne peut avoir au plus un seul état local actif, alors, elles ne peuvent pas être activées en parallèle. De plus, pour chaque automate a, une et une seule transition locale est choisie pour être activée parmi ses $\mathcal{T}_a$ jouables. Ainsi, dans un même état global, les transitions locales concurrentes doivent être activées séparément et chacune change l'état local actif de l'automate ciblé vers l'état local qu'elle a comme destination. Et donc elles font évoluer le réseau vers des états globaux distincts. Par conséquent, un non-déterminisme apparaît dans le graphe d'états.

Par exemple, le modèle AN de la figure 5.1 en page 134 présente deux transitions locales concurrentes dans l'état global $\langle a_1, b_0, c_1, d_0\rangle$: $b_0 \xrightarrow{\{d_0\}} b_1$ et $b_0 \xrightarrow{\{a_1, c_1\}} b_2$. Ces deux transitions locales sont jouables dans les transitions globales représentées par les flèches rouges dans la figure 5.5 en page 147 : $\langle a_1, b_0, c_1, d_0\rangle \rightarrow_{U^{\mathrm{syn}}} \langle a_1, b_1, C_1, d_0\rangle$ et $\langle a_1, b_0, c_1, d_0\rangle \rightarrow_{U^{\mathrm{syn}}} \langle a_1, b_2, c_1, d_0\rangle$. Il faut noter que ce non-déterminisme existe également dans le graphe d'états calculé selon la sémantique asynchrone : les transitions globales sont aussi représentées par des flèches rouges dans la figure 5.4 en page 146.

On peut noter aussi que si tous les automates contiennent seulement deux états locaux (un tel AN est souvent appelé "AN Booléen"), la sémantique synchrone devient alors complètement déterministe. En effet, il n'est plus possible de trouver des transitions locales concurrentes car pour chaque origine d'une transition locale, il ne peut y avoir qu'une seule destination. Ceci est dû au fait que l'origine et la destination d'une transition locale doivent être différentes. Alors pour toutes les transitions locales d'un automate a, il n'y a toujours qu'une seule destination et un seul origine : de a_0 vers a_1 ou à l'inverse. Et dans une dynamique qui suit une sémantique synchrone, si cette transition locale est jouable, alors elle sera surement activée. Ce qui implique qu'à partir d'un état global donné il n'existe toujours qu'un seul chemin possible. Cette observation peut accélérer les calculs dans ce cas particulier car il n'y a toujours qu'un seul chemin possible.

5.3 Vérification de l'atteignabilité

L'objectif de cette section est de définir la problématique de l'*atteignabilté* dans les AN.

5.3.1 Définition de la problématique de l'atteignabilité

La problématique de l'atteignabilité dans les AN consiste à vérifier l'existence ou pas d'un chemin activé depuis un état global initialement donné et qui permet d'atteindre un ou plusieurs états locaux fixés comme objectif. Ceci peut se résumer par la question suivante : "Étant donné un état global initial ζ et un ensemble d'états locaux ω donnés comme objectifs, existe-t-il un chemin H activé depuis l'état global ζ, compatible avec la dynamique du modèle AN selon une sémantique donnée de mise à jour de la dynamique, tel que H permet d'atteindre un état global dans lequel tous les états locaux de ω sont présents simultanément ?"

Formellement, l'atteignabilité des états locaux de différents automates à partir d'un état global initial ζ se traduit par l'existence d'un chemin H, c'est-à-dire d'une séquence d'états globaux tel que son état global initial est ζ ($H_0 = \zeta$) et qu'il y a un état global $H_n \in H$ dans lequel tous les états locaux objectifs sont actifs.

Définition 5.10 (Atteignabilité)**.** Soient $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$ un réseau d'automates, et $\mathbf{LS} = \cup_{a\in\Sigma}\mathcal{S}_a$ l'ensemble de tous les états locaux de $\mathcal{AN}$. Soit $\omega \in \wp(\mathbf{LS})$ un ensemble d'états locaux fixés comme objectifs et $\zeta \in \mathcal{S}$ un état global de $\mathcal{AN}$.
ω est *atteignable* à partir de ζ si et seulement s'il existe un chemin $H = (H_i)_{i\in[\![0;n]\!]} \in \mathcal{S}^{n+1}$ avec $n \in \mathbb{N}^*$ et $H_0 = \zeta$ tel que $\forall a \in \Sigma$, si $\mathcal{S}_a \cap \omega = a_j$ alors $H_n[a] = a_j$.

Le chemin recherché H active successivement les états locaux fixés comme objectif dans ω. Et notre but est de développer une méthode qui vérifie si ce chemin existe ou pas. En effet, son existence implique une réponse positive à la question de l'atteignabilité.

Par exemple, nous donnons dans les figures 5.2 et 5.3 ci-dessous, des illustrations des évolutions possibles du modèle AN de la figure 5.1 en page 134 selon respectivement la sémantique asynchrone et synchrone de la dynamique. Les deux figures montrent des chemins possibles pour l'activation de l'objectif $\omega = \{d_2\}$ à partir de l'état global initial $\zeta = \langle a_1, b_2, c_0, d_1\rangle$.

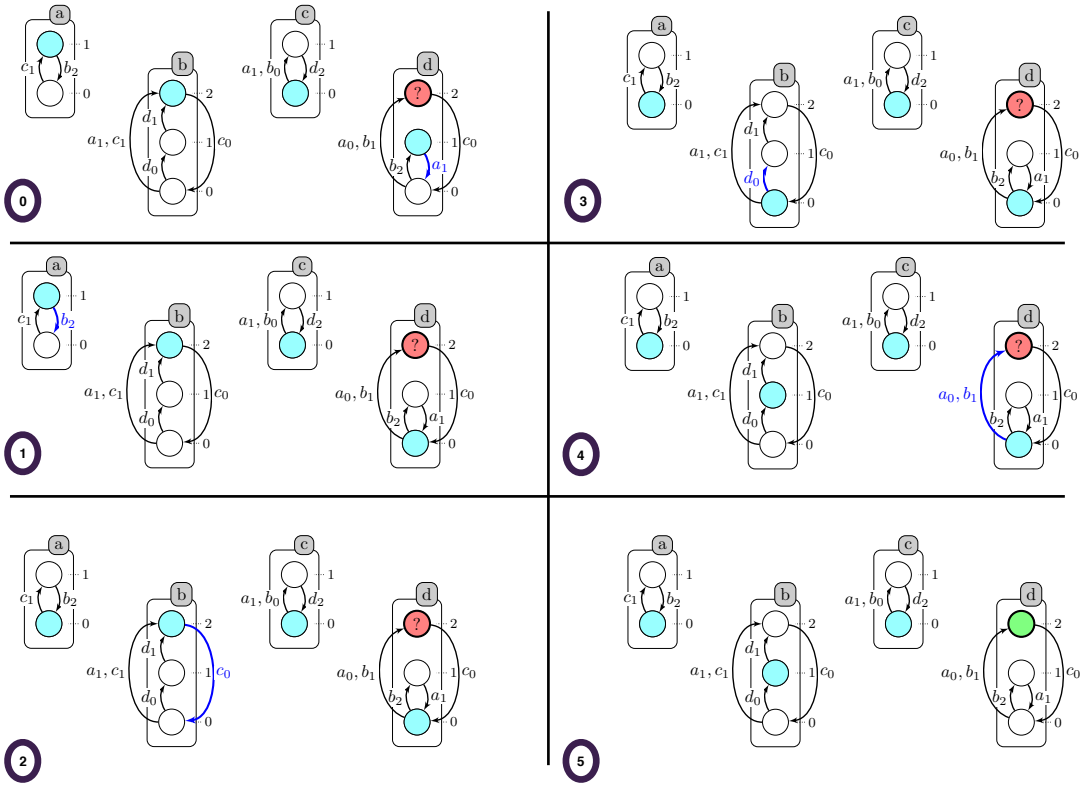


Figure 5.2 : Illustration d'une évolution dynamique selon la sémantique **asynchrone** d'un réseau d'automates à partir de l'état global initial $\langle a_1, b_2, c_0, d_1 \rangle$ jusqu'à l'activation de l'objectif d_2. L'état local d_2 est coloré en rouge au cours de l'évolution du réseau (inactif) et en vert quand il est atteint (actif). A chaque instant, les états locaux actifs des automates du réseau sont colorés en bleu clair et les transitions locales activées sont colorées en bleu.

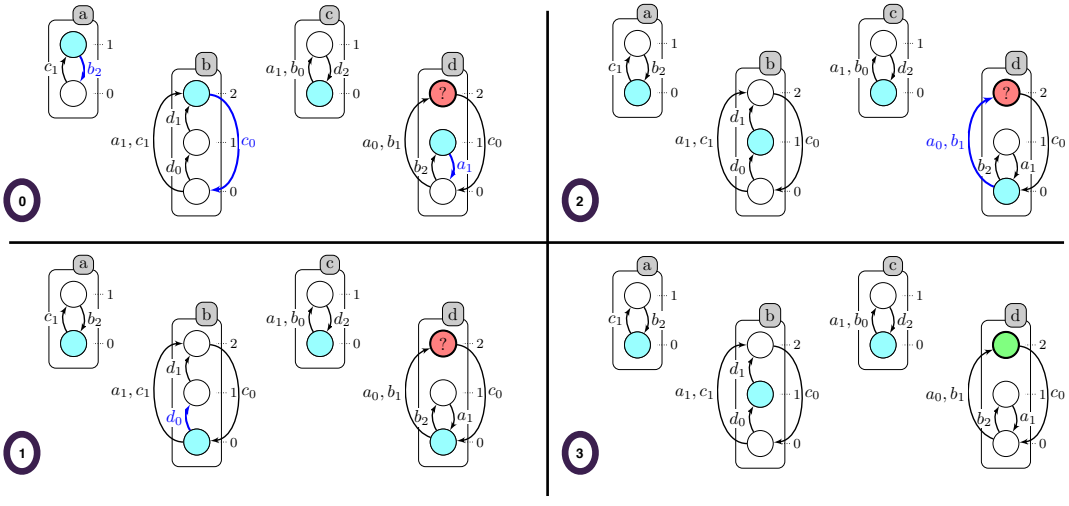


Figure 5.3 : Illustration d'une évolution dynamique selon la sémantique **synchrone** d'un réseau d'automates à partir de l'état global initial $\langle a_1, b_2, c_0, d_1 \rangle$ jusqu'à l'activation de l'objectif d_2. L'état local d_2 est coloré en rouge au cours de l'évolution du réseau (inactif) et en vert quand il est atteint (actif). A chaque instant, les états locaux actifs des automates du réseau sont colorés en bleu clair et les transitions locales activées sont colorées en bleu.

Le chemin H qui permet d'atteindre d_2 dans la figure 5.2 est trouvé selon **la sémantique asynchrone** ; dans chaque état global, une seule transition locale est activée parmi celles qui sont jouables (voir définition 5.3 en page 136). Cette évolution dynamique du AN correspond au chemin suivant :
$H = (\langle a_1, b_2, c_0, d_1\rangle ; \langle a_1, b_2, c_0, d_0\rangle ; \langle a_0, b_2, c_0, d_0\rangle ; \langle a_0, b_0, c_0, d_0\rangle ; \langle a_0, b_1, c_0, d_0\rangle ; \langle a_0, b_1, c_0, d_2\rangle)$.
Ainsi, H est de longueur $n = 5$. On peut voir que tous les états globaux parcourus par H ne sont visités qu'une seule fois. Ainsi, $\mathsf{sr}(H) = \emptyset$ et donc la taille de la trace de H est égale à : $|\mathsf{trace}(H)| = n + 1 - |\mathsf{sr}(H)| = 5 + 1 - 0 = 6$. Donc au total, 6 états globaux sont parcourus par H. Ils peuvent être visualisés dans la figure 5.4 en page 146 : l'état global initial de H, $\langle a_1, b_2, c_0, d_1\rangle$, est coloré en vert et l'état final auquel d_2 est atteint, $\langle a_0, b_1, c_0, d_2\rangle$, est coloré en bleu.

La figure 5.3 ci-dessus illustre l'évolution de la dynamique du même modèle AN jusqu'à l'activation de d_2 mais selon une sémantique différente qui est **la sémantique synchrone**. Elle correspond au chemin : $H' = (\langle a_1, b_2, c_0, d_1\rangle ; \langle a_0, b_0, c_0, d_0\rangle ; \langle a_0, b_1, c_0, d_0\rangle ; \langle a_0, b_1, c_0, d_2\rangle)$. H' est alors de longueur $n = 3$ et n'a pas de répétitions d'états globaux, donc la taille de sa trace est égale à 4 : $|\mathsf{trace}(H')| = 4$.

Nous notons que la définition 5.10 concernant la propriété de l'atteignabilité, est valable quelle que soit la sémantique de la dynamique (asynchrone ou synchrone ou autre).

La vérification de la propriété de l'atteignabilité peut être confondue avec la vérification des propriétés de la logique temporelle (logique CTL) sous la forme **EF** en *model checking*. En effet, la logique temporelle est un domaine mathématique qui s'intéresse à l'évolution des variables dans le temps, et la vérification de l'atteignabilité sert à vérifier si au cours de l'évolution dans le temps, des variables du modèle (les automates en AN) atteignent des états précis (états locaux).

Nous pouvons citer quelques travaux qui manipulent les modèles discrets et qui ont été développés pour vérifier la propriété dynamique de l'atteignabilité dans les RRB. Par exemple, la bibliothèque libDDD (bibliothèque des diagrammes de décision de données) pour la vérification symbolique des propriétés CTL & LTL (LIP6/Move, libDDD; Thierry-Mieg, Poitrenaud, Hamez & Kordon, 2009) et qui n'est pas spécifiquement consacrée aux RRB. Elle peut notamment être utilisée pour vérifier la propriété d'atteignabilité dans les réseaux de Petri.

Un avantage de ces analyses exhaustives des modèles est qu'elles permettent d'exprimer plusieurs types de comportements. Ainsi elles ont potentiellement un grand champ d'application autre qu'en biologie. Cependant, ces outils calculent le graphe d'états complet correspondant à un RRB, ce qui nécessite un espace mémoire énorme et ils ne sont donc pas toujours en mesure de vérifier les propriétés dynamiques des grands réseaux.

Plusieurs autres techniques de modélisation et d'analyse prometteuses ont été développées en utilisant des réseaux booléens, des réseaux de Petri (Heljanko & Niemelä, 2001) ou des réseaux bayésiens (Numata, Imoto & Miyano, 2008) en essayant d'optimiser le temps de calcul ainsi que le résultat (Paoletti et al., 2014).

La méthode d'analyse que nous développons dans cette section est basée sur une approche d"analyse dynamique exhaustive. Une particularité est son implémentation en programmation logique : avec le langage ASP. Nous présentons son algorithme dans la section suivante.

5.3.2 Idée intuitive pour la résolution de la problématique de l'atteignabilité

L'algorithme 3 ci-dessous suit l'idée intuitive de la méthode implémentée en ASP pour la résolution de la problématique de l'atteignabilité. Nous notons que cette méthode est inspirée de la définition 5.10 en page 142 ; à partir d'un état global initial ζ, vérifier s'il existe au moins un chemin de longueur inférieure ou égale à $n \in \mathbb{N}^*$, tel qu'il atteint un état global dans lequel tous les états locaux de ω sont actifs. Cet entier strictement positif n est la longueur maximale fixée du chemin qui permet d'atteindre les objectifs. Ainsi, on peut considérer que la recherche est *bornée* par un entier n déterminé arbitrairement.

Algorithm 3 : verifAtteignabilite$(\mathcal{AN}, \zeta, \omega, U, n)$

Entrée :

$\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$: un AN
$\zeta \in \mathcal{S}$: un état global initial
$\omega \in \wp(\mathbf{LS})$: un ensemble d'états locaux objectifs avec $\mathbf{LS} = \cup_{a \in \Sigma} \mathcal{S}_a$
$U \in \{U^{\text{asyn}}, U^{\text{syn}}\}$
$n \in \mathbb{N}^*$: la longueur maximale du chemin qui atteint tous les états de ω

Sortie :

$verif$: variable booléenne qui est vraie si tous les éléments de ω sont atteints.

Début :

$verif \leftarrow faux$ *// initialisation*
$k \leftarrow 1$
tant que $(k < n \wedge verif = faux)$ **faire**
  si $(\exists H = (H_i)_{i \in [\![0;k]\!]} \in \mathcal{S}^{k+1}$ tel que $H_0 = \zeta \wedge \forall a_i \in \omega,\ H_k[a] = a_i)$ **alors**
    $verif \leftarrow vrai$ *// tous les éléments de ω sont atteints dans H_k*
  fin si
  $k \leftarrow k + 1$
fin tant que
retourner $verif$

L'algorithme 3 ci-dessus peut être assimilé à une recherche en largeur dans un graphe d'états d'un AN. En effet, pour chaque valeur de k (qui est égale à la longueur du chemin calculé) la méthode vérifie s'il existe un état global atteint et dans lequel tous les éléments objectifs (de ω) sont atteints. Si c'est le cas le calcul s'arrête et le résultat de l'algorithme est vrai. Sinon la vérification est refaite avec un chemin de longueur égale à $k + 1$.

Nous détaillons dans la section 6.4 du chapitre 6 en page 186, les programmes logiques de l'implémentation de cet algorithme en ASP. De plus, nous présentons les programmes logiques qui calculent la dynamique des AN selon **la sémantique asynchrone** d'une part et selon **la sémantique synchrone** d'autre part. En outre, nous y présentons le programme ASP qui calcule la dynamique des réseaux d'automates avec le temps (T-AN) dont la sémantique est introduite dans le chapitre 3. Nous présentons aussi dans le chapitre 6 les résultats obtenus par cette méthode sur des réseaux d'applications réelles.

Souvent, l'évolution dynamique des systèmes biologiques finit par atteindre un domaine

piège : un ensemble d'états globaux à partir desquels le système ne peut pas échapper. Un domaine piège minimal est appelé un attracteur. Les attracteurs présentent un intérêt particulier dans la modélisation des RRB. En effet, la dynamique de chaque RRB est caractérisée par un ensemble d'attracteurs. Ainsi, pour qu'un AN modélise correctement un RRB, il doit présenter les attracteurs qui sont cohérents avec ceux qui sont observés dans le système biologique que le AN modélise. Par conséquent, dans le but d'identifier les attracteurs dans un AN, nous l'étudions formellement dans la section suivante.

5.4 Identification des attracteurs

Les *points fixes* qui sont des attracteurs singletons (aussi appelés *états stables*) et les *attracteurs cycliques* qui sont des attracteurs non singletons sont les deux structures à long terme que la dynamique finit souvent par atteindre et desquels il n'est plus possible de s'échapper. En effet, tout chemin de longueur maximale d'un réseau converge finalement vers un attracteur. De telles structures présentent une caractérisation importante du RRB modélisé. En effet, l'identification des attracteurs est l'un des processus de la validation après l'apprentissage des modèles.

Dans cette section, nous nous concentrons sur la problématique de l'identification des attracteurs dans les RRB modélisés en AN. Dans les figures 5.4 et 5.5 ci-dessous, nous présentons des parties des graphes d'états calculées respectivement selon les sémantiques asynchrone et synchrone de la dynamique pour le modèle AN de la figure 5.1 en page 134.

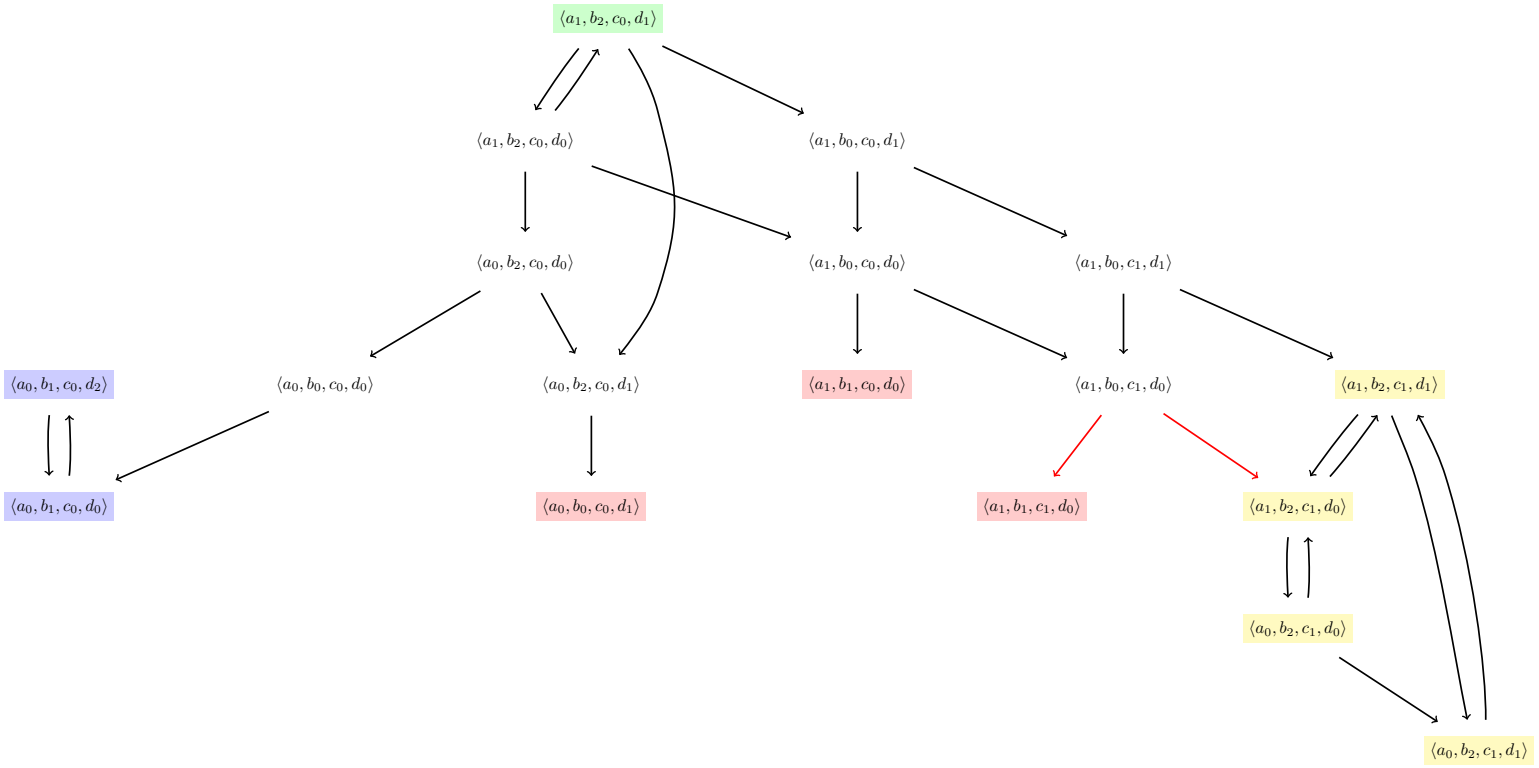


Figure 5.4 : Une partie du graphe d'états du réseau d'automates de la figure 5.1 en page 134 calculé selon la sémantique **asynchrone** à partir de l'état global initial $\langle a_1, b_2, c_0, d_1\rangle$ (en vert) jusqu'à l'atteinte des attracteurs. On observe 3 points fixes (en rouge), un attracteur de taille 2 (en bleu) et un attracteur de taille 4 (en jaune).

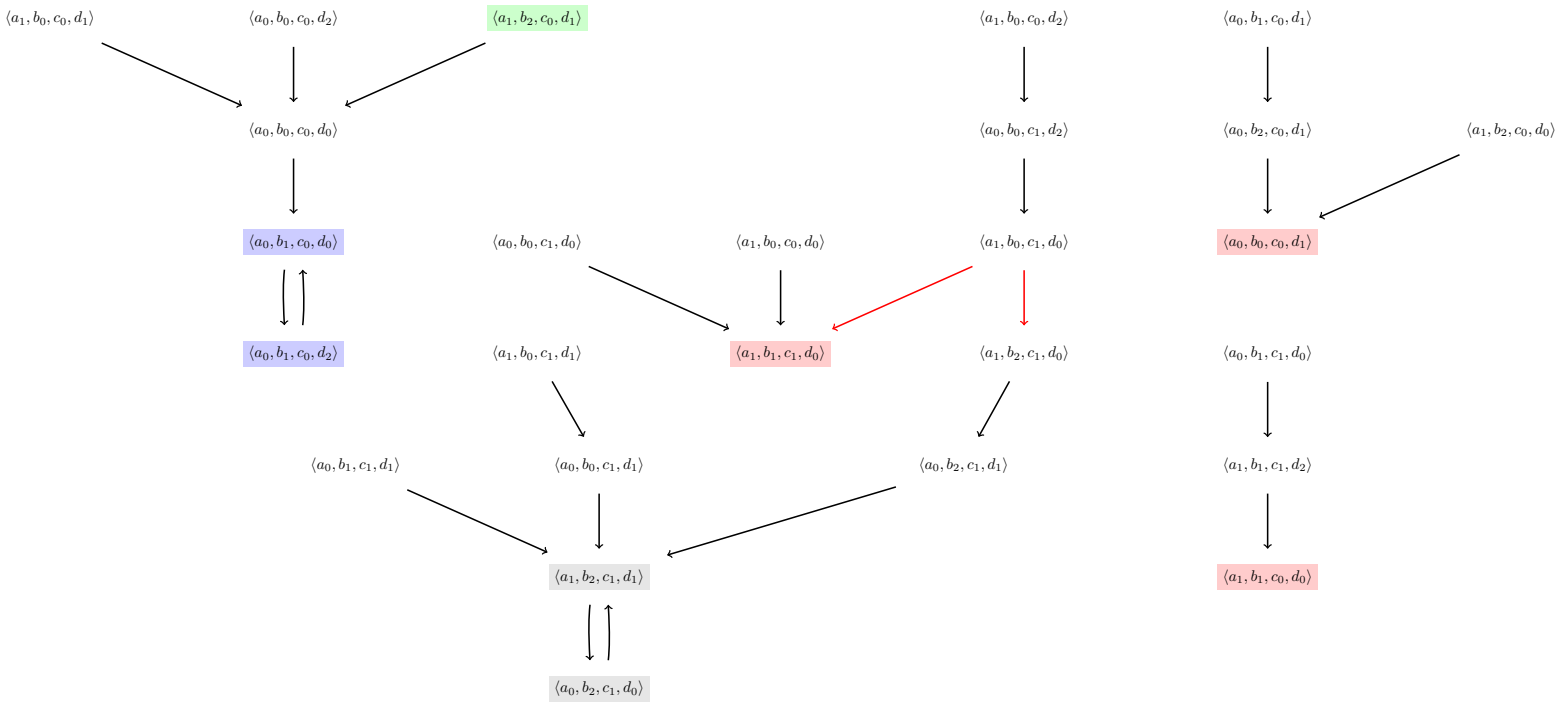


Figure 5.5 : Une partie du graphe d'états de transitions du réseau d'automates de la figure 5.1 en page 134 calculé selon la sémantique **synchrone** à partir de plusieurs états globaux initiaux comme $\langle a_1, b_2, c_0, d_1\rangle$ (en vert). On observe 3 points fixes (en rouge), et deux attracteurs de taille 2 (en bleu et en gris).

Les états globaux colorés en rouges représentent les points fixes qui sont des états globaux singletons terminaux de la dynamiques du modèle (ils n'ont pas de successeurs). On peut remarquer qu'ils sont les mêmes dans les deux sémantiques (synchrone et asynchrone) : $\langle a_1, b_1, c_1, d_0\rangle$, $\langle a_1, b_1, c_0, d_0\rangle$ et $\langle a_0, b_0, c_0, d_1\rangle$.

Les ensembles des états globaux en bleu, en jaune et en gris présentent des attracteurs cycliques (non singletons) et qui sont aussi des états globaux terminaux. Dans la figure 5.4, on trouve un attracteur de taille 2 (en bleu) : $\{\langle a_0, b_1, c_0, d_0\rangle, \langle a_0, b_1, c_0, d_2\rangle\}$ et un attracteur de taille 4 (en jaune) : $\{\langle a_1, b_2, c_1, d_1\rangle, \langle a_0, b_2, c_1, d_1\rangle, \langle a_0, b_2, c_1, d_0\rangle, \langle a_1, b_2, c_1, d_0\rangle\}$. Dans la figure 5.4, on trouve deux attracteurs de taille 2 : $\{\langle a_0, b_1, c_0, d_0\rangle, \langle a_0, b_1, c_0, d_2\rangle\}$ (en bleu) et $\{\langle a_1, b_2, c_1, d_1\rangle, \langle a_0, b_2, c_1, d_0\rangle\}$ (en gris).

Dans la suite de cette section, nous introduisons comment identifier formellement les attracteurs dans un AN. En effet, les attracteurs ont des caractéristiques dynamiques bien précises. Ainsi, nous séparons cette étude en deux parties : une première sur les attracteurs non-singletons (appelés *les attracteurs cycliques*) et une deuxième partie sur les attracteurs singletons (appelés les *points fixes*).

5.4.1 Les attracteurs cycliques

Nous étudions dans cette section l'attracteur cyclique (un ensemble non-singleton d'états globaux) qui est une fois atteint par la dynamique, le système ne peut plus en échapper. En effet, dans de telles structures, le système cycle indéfiniment sans en sortir. D'une façon générale, les attracteurs sont englobés par des ensembles plus larges appelés des *domaines pièges* (définiton 5.11 ci-dessous).

Définition 5.11 (Domaine piège)**.** Soient $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$ un réseau d'automates et U une sémantique de la dynamique. L'ensemble des états globaux $\mathbf{T} \subseteq \mathcal{S}$ est appelé un *domaine piège* (selon la sémantique U) si et seulement si tout successeur d'un état global de $\mathbf{T}$ appartient aussi à $\mathbf{T}$:

$$\mathbf{T} \text{ est un domaine piège } \Leftrightarrow \forall \zeta_1 \in \mathbf{T} \wedge \forall \zeta_2 \in \mathcal{S} \text{ si } \exists \zeta_1 \rightarrow_U \zeta_2 \text{ alors } \zeta_2 \in \mathbf{T}.$$

Exemple. Prenons le graphe d'états de la figure 5.4 en page 146. Selon la sémantique synchrone, les ensembles des états globaux suivants sont des domaines pièges pour le modèle AN de la figure 5.1 en page 134 :

- $\mathbf{T} = \{\langle a_0, b_0, c_0, d_0\rangle, \langle a_0, b_1, c_0, d_0\rangle, \langle a_0, b_1, c_0, d_2\rangle\}$;
- $\mathbf{T}' = \{\langle a_0, b_2, c_0, d_1\rangle, \langle a_0, b_0, c_0, d_1\rangle\}$;
- $\mathbf{T}'' = \{\langle a_0, b_2, c_1, d_0\rangle, \langle a_0, b_2, c_1, d_1\rangle, \langle a_1, b_2, c_1, d_1\rangle, \langle a_1, b_2, c_1, d_0\rangle\}$.

On peut aussi donner un contre-exemple d'un ensemble d'états globaux qui n'est pas un domaine piège : $\mathbf{E} = \{\langle a_1, b_0, c_0, d_0\rangle, \langle a_1, b_1, c_0, d_0\rangle\}$ (voir figure 5.4 en page 146 où $\langle a_1, b_1, c_0, d_0\rangle$ est coloré en rouge). En effet, il existe une transition globale qui fait évoluer la dynamique du réseau à partir d'un état de $\mathbf{E}$ vers d'autres états globaux qui n'appartiennent pas à $\mathbf{E}$. C'est la transition globale : $\langle a_1, b_0, c_0, d_0\rangle \rightarrow_{U^{\text{asyn}}} \langle a_1, b_0, c_1, d_0\rangle$ tel que $\langle a_1, b_0, c_1, d_0\rangle \notin \mathbf{E}$. En plus, à partir de $\langle a_1, b_0, c_1, d_0\rangle$ le réseau continue à évoluer vers d'autres états globaux. Ainsi, la dynamique du réseau peut échapper de $\mathbf{E}$. Et donc $\mathbf{E}$ n'est pas un domaine piège.

Remarque. Si un état global $\zeta \in \mathbf{T}$, avec $\mathbf{T}$ un domaine piège, alors ζ est un état *absorbant*. En effet, au moment où la dynamique atteint un état absorbant, il est certain qu'elle ne va plus sortir du domaine piège auquel il appartient. Ainsi, si un état global n'est pas absorbant, alors il est *transitoire*.

Dans ce travail, nous nous concentrons plus précisément sur les attracteurs qui sont des domaines pièges *minimaux* pour l'inclusion (définition 5.12 ci-dessous). Cette notion de minimalité est expliquée par le fait que si un ensemble d'états globaux est identifié comme étant un attracteur, alors aucun de ses sous-ensembles n'est un attracteur (qu'il soit singleton ou non-singleton).

Définition 5.12 (Attracteur)**.** Soient $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$ un réseau d'automates et U une sémantique de la dynamique. L'ensemble des états globaux $\mathbf{A} \subset \mathcal{S}$ est appelé un *attracteur* (selon la sémantique U) si et seulement si $\mathbf{A}$ est un domaine piège minimal pour l'inclusion.

Exemple. Parmi les 3 domaines pièges de l'exemple précédent en page 139, seul $\mathbf{T}''$ est un attracteur. En effet, $\mathbf{T}$ et $\mathbf{T}'$ contiennent des attracteurs dont les tailles sont inférieures à leur taille ; $\mathbf{T}$ contient un attracteur de taille 2 (coloré en bleu dans la figure 5.4 en page 146) : $\{\langle a_0, b_1, c_0, d_0\rangle, \langle a_0, b_1, c_0, d_2\rangle\}$ et $\mathbf{T}'$ contient un attracteur (singleton) de taille 1 : $\{\langle a_0, b_0, c_0, d_1\rangle\}$ (coloré en rouge).

Ainsi, une caractéristique très importante sur les attracteurs peut être déduite. On remarque qu'à partir de chaque état global, tous ses autres états globaux sont (directement ou indirectement) atteints. Ainsi, on peut dire que les attracteurs non-singletons sont nécessairement *cycliques*. Autrement dit, tout chemin qui parcourt tous les états globaux d'un attracteur non singleton est nécessairement un *cycle*. D'où leur nom "*attracteurs cycliques*".

Ainsi, afin d'étudier formellement les attracteurs cycliques d'un AN nous introduisons tout d'abord la notion de cycle dans la définition 5.13 ci-dessous. Comme mentionné informellement et brièvement précédemment, un cycle n'est qu'un chemin en boucle : le premier état global dans la séquence du chemin est identique au dernier.

Définition 5.13 (Cycle)**.** Soient $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$ un réseau d'automates et U une sémantique de mise à jour de la dynamique et $\mathbf{C} = (C_i)_{i\in[\![0;n]\!]} \in \mathcal{S}^{n+1}$ un chemin de longueur $n \in \mathbb{N}^*$ trouvé selon la sémantique U. $\mathbf{C}$ est appelé un *cycle* de longueur n si et seulement si ce chemin $\mathbf{C}$ retourne à son état global initial :

$$\mathbf{C}_0 = \mathbf{C}_n.$$

Exemple. Reprenons le chemin H donné dans l'exemple en page 139 et qui est de longueur $n = 6$:

$$H = (\langle a_1, b_2, c_1, d_1\rangle ; \langle a_0, b_2, c_1, d_1\rangle ; \langle a_1, b_2, c_1, d_1\rangle ; \langle a_1, b_2, c_1, d_0\rangle ;$$
$$\langle a_0, b_2, c_1, d_0\rangle ; \langle a_0, b_2, c_1, d_1\rangle ; \langle a_1, b_2, c_1, d_1\rangle).$$

H est un cycle parce que $H_6 = H_0$. Ainsi, H peut être aussi noté $\mathbf{C}$.

Nous montrons dans le lemme 5.2 ci-dessous que la trace de tout cycle (quelle que soit la sémantique de la dynamique) est une *composante fortement connexe*. En effet, un cycle permet de "boucler" entre tous les états globaux qu'il contient, et donc chaque état global visité par ce cycle est atteignable par tous les autres états globaux visités par ce cycle. Rappelons que si un état global est visité par un cycle (ou chemin) alors il appartient à la trace de ce cycle (définition 5.7 en page 139).

Lemme 5.2 (La trace d'un cycle est une composante fortement connexe)**.** *Les traces des cycles sont exactement des composantes fortement connexes (selon une sémantique donnée de la dynamique).*

Démonstration. ($\Rightarrow$) À partir de n'importe quel état global d'un cycle, il est possible d'atteindre tous ses autres états globaux. Par conséquent, la trace de ce cycle est une composante fortement connexe. ($\Leftarrow$) Soit $\mathbf{S} = \{\zeta_i\}_{i\in[\![0;n]\!]}$ une composante fortement connexe, donc pour tout $i \in [\![0\,;n]\!]$, il existe un chemin H^i construit à partir des éléments de $\mathbf{S}$, avec $H^i_0 = \zeta_i$ et $H^i_{|H^i|} = \zeta_{i+1}$ tel que ζ_{i+1} est un successeur de ζ_i et avec $H^n_{|H^n|} = \zeta_0$ pour $i = n$.
Nous créons un chemin $\mathbf{C}$ par concaténation de tous les chemins $H^0, H^1, \ldots, H^n$ en fusionnant tous les chemins successifs par le remplacement du dernier élément de chacun par le premier de l'autre car ils sont identiques : $\forall i \in [\![0\,;n-1]\!], H^i_{|H^i|} = \zeta_{i+1} = H^{i+1}_0$.
Ainsi, $\mathbf{C}$ est un cycle, parce que $\mathbf{C}_0 = H^0_0 = \zeta_0 = H^n_{|H^n|} = \mathbf{C}_{|\mathbf{C}|}$. En outre, $\forall i \in [\![0\,;n]\!], \zeta_i = H^i_0 \in \mathsf{trace}(\mathbf{C})$, ainsi $\mathbf{S} \subseteq \mathsf{trace}(\mathbf{C})$. Finalement, puisque seulement les états globaux de $\mathbf{S}$ sont utilisés pour construire $\mathbf{C}$, alors $\mathsf{trace}(\mathbf{C}) \subseteq \mathbf{S}$. Par conséquent, $\mathsf{trace}(\mathbf{C}) = \mathbf{S}$. □

Le lemme 5.2 ci-dessus, permet de déduire qu'à partir de tout ensemble d'états globaux ayant un chemin les reliant et qui forme une composante fortement connexe, alors un cycle peut être trouvé. Donc, cette équivalence entre la trace d'un cycle et la composante fortement connexe nous permet de donner une définition alternative pour un attracteur (lemme 5.3 ci-dessous).

En effet, dans la définition 5.12 en page 148, les attracteurs sont définis d'une façon classique en étant des domaines pièges minimaux. Par contre, on propose une identification

alternative des attracteurs qui nous facilite son implémentation en programmation logique (présentée dans le chapitre 6). Le lemme 5.3 ci-dessous, présente qu'alternativement, un attracteur peut être défini en étant un domaine piège cyclique (ou juste un *cycle piège*). En d'autres termes, l'exigence de la minimalité est remplacée par une exigence de cyclicité.

Lemme 5.3 (Attracteurs en tant que traces des cycles pièges)**.** *Les attracteurs sont exactement les traces des cycles qui sont des domaines pièges.*

Démonstration. ($\Rightarrow$) Par définition 5.12, un attracteur est un domaine piège. Il est aussi une composante fortement connexe, et donc, selon le lemme 5.2 ci-dessus, un attracteur est la trace d'un cycle. ($\Leftarrow$) Soit $\mathbf{C}$ à la fois un cycle et un domaine piège. Selon le lemme 5.2, $\mathbf{C}$ est aussi une composante fortement connexe. Prouvons par contradiction que $\mathbf{C}$ est un domaine piège minimal, en supposant que $\mathbf{C}$ n'est pas minimal. Ce qui signifie qu'il existe un domaine piège plus petit $\mathbf{D} \subsetneq \mathbf{C}$. Considérant $x \in \mathbf{D}$ et $y \in \mathbf{C} \setminus \mathbf{D}$. Puisque $\mathbf{D}$ est un domaine piège, il n'existe alors pas de chemin entre x et y ; ce qui est contradictoire avec le fait que $\mathbf{C}$ soit une composante fortement connexe. En effet x et y appartiennent à $\mathbf{C}$ alors il existe forcément un chemin entre x et y. Par conséquent, $\mathbf{C}$ est un domaine piège minimal, et ainsi sa trace est un attracteur. □

Exemple. Les graphes d'états des figures 5.4 et 5.5 en pages 146-147 présentent différents attracteurs :

- $\{\langle a_0, b_1, c_0, d_0\rangle, \langle a_0, b_1, c_0, d_2\rangle\}$ est coloré en bleu et apparaît sur les deux figures. C'est un attracteur cyclique : le cycle qui visite tous ses états globaux est de longueur minimale $n = 2$.
- $\{\langle a_0, b_2, c_1, d_0\rangle, \langle a_0, b_2, c_1, d_1\rangle, \langle a_1, b_2, c_1, d_1\rangle, \langle a_1, b_2, c_1, d_0\rangle\}$ ne se trouve que pour la sémantique asynchrone et est coloré en jaune sur la figure 5.4. C'est un attracteur cyclique : le cycle qui visite tous ses états globaux est de longueur minimale $n = 4$.
- $\{\langle a_1, b_2, c_1, d_1\rangle, \langle a_0, b_2, c_1, d_0\rangle\}$ est, au contraire, présent uniquement pour la sémantique synchrone et est coloré en gris sur la figure 5.5. C'est aussi un attracteur cyclique : le cycle qui visite tous ses états globaux est de longueur minimale $n = 2$.

En l'occurrence, on donne un exemple de cycle de longueur $n = 4$ qui visite tous les états globaux du deuxième attracteur :

$$\mathbf{C} = (\langle a_0, b_2, c_1, d_0\rangle ; \langle a_0, b_2, c_1, d_1\rangle ; \langle a_1, b_2, c_1, d_1\rangle ; \langle a_1, b_2, c_1, d_0\rangle ; \langle a_0, b_2, c_1, d_0\rangle).$$

L'absence de flèches sortantes à partir de tous les états globaux de $\mathsf{trace}(\mathbf{C})$ (qui sont colorés en jaunes dans la figure 5.4 en page 146) vers des états globaux en dehors de $\mathsf{trace}(\mathbf{C})$, confirme le fait que ce dernier présente un domaine piège. Il s'agit donc d'un cycle piège.

Propriété 5.1. *La trace d'un cycle de longueur $n \in \mathbb{N}^*$ qui est un domaine piège et qui ne contient pas de cycles de longueur strictement inférieure à n est un attracteur de taille n.*

Démonstration. Soit $\mathbf{C}$ un cycle piège de longueur $n \in \mathbb{N}^*$. Si un cycle $\mathbf{C}$ ne contient pas d'autres cycles de longueur strictement inférieure à sa longueur n, alors il n'y a pas d'états globaux répétés, c'est-à-dire visités plus qu'une fois par $\mathbf{C}$ (sauf le premier et le dernier éléments sont identiques). $\mathbf{C}$ étant aussi un chemin, on a alors, selon la définition

5.8 en page 140, $\mathsf{sr}(\mathbf{C}) = \{n\}$, donc $|\mathsf{sr}(\mathbf{C})| = 1$. Dans le lemme 5.1 en page 140, nous avons montré que la taille de la trace d'un chemin H de longueur n est égale à $|\mathsf{trace}(H)| = n + 1 - |\mathsf{sr}(H)|$, ainsi en appliquant cette formule pour le cycle $\mathbf{C}$, on trouve que $|\mathsf{trace}(\mathbf{C})| = n + 1 - |\mathsf{sr}(\mathbf{C})| = n$. □

Comme expliqué précédemment, le lemme 5.3 ci-dessus est utilisé pour la recherche des attracteurs dans la section 6.5 en page 200 du chapitre 6. En effet, la recherche directe de domaines pièges minimaux serait trop lente et consomme beaucoup de mémoire. Dans le but d'éviter ces problèmes, nous énumérons des cycles de longueur $n \in \mathbb{N}^*$ selon une sémantique donnée, et puis nous filtrons ceux qui ne sont pas des domaines pièges. Les traces des cycles restants sont ainsi des attracteurs formés de cycles de longueur n. Puisque le lemme 5.3 précédent prouve que les traces des cycles pièges sont des attracteurs, alors il assure la complétude et l'exhaustivité d'une telle méthode de recherche d'attracteurs. Nous introduisons plus de détails cette méthode et son algorithme dans la section 5.4.4 en page 156. Mais avant, nous continuons à étudier les propriétés formelles des attracteurs et qui nous facilitent la compréhension de la démarche de la méthode de la recherche des attracteurs.

5.4.2 Étude des caractéristiques dynamiques des attracteurs cycliques

Nous avons montré dans la section précédente que les attracteurs qui sont des domaines pièges minimaux ne peuvent être que des composantes fortement connexes. Autrement dit, tout attracteur non singleton est cyclique. Par contre, il est à noter que toute trace d'un cycle qui n'est pas un piège ne présente pas nécessairement un attracteur. En effet, il suffit d'avoir une transition transitoire à partir de l'un des ses états globaux vers un autre état global qui n'y appartient pas (voir lemme 5.4).

Lemme 5.4 (Caractérisation des non-attracteurs)**.** *Soit* $\mathbf{E} \subseteq \mathcal{S}$ *un ensemble d'états globaux. Si* $\exists \zeta_1 \in \mathbf{E}$ *et* $\exists \zeta_2 \in \mathcal{S} \setminus \mathbf{E}$ *tel que* $\zeta_1 \rightarrow_U \zeta_2$ *alors* $\mathbf{E}$ *n'est pas un attracteur.*

Démonstration. Si $\exists \zeta_1 \in \mathbf{E}$ et $\exists \zeta_2 \in \mathcal{S} \setminus \mathbf{E}$ tel que $\zeta_1 \rightarrow_U \zeta_2$ alors $\mathbf{E}$ n'est pas un domaine piège (voir définition 5.11 en page 148) et ainsi $\mathbf{E}$ n'est pas un attracteur (voir définition 5.12 en page 148). □

Exemple. Par exemple, dans la figure 5.4 en page 146, on peut distinguer que le chemin suivant est un cycle sans que sa trace soit un attracteur :

> $\mathbf{C} = (\langle a_1, b_2, c_0, d_1\rangle ; \langle a_1, b_2, c_0, d_0\rangle ; \langle a_1, b_2, c_0, d_1\rangle)$ est un cycle de longueur $n = 2$ et sa trace $\mathsf{trace}(\mathbf{C}) = \{\langle a_1, b_2, c_0, d_1\rangle, \langle a_1, b_2, c_0, d_0\rangle\}$ n'est pas un attracteur.

En effet, comme c'est visible sur la figure 5.4 en page 146, ces deux états globaux ont des transitions globales sortantes de $\mathsf{trace}(\mathbf{C})$. Par exemple, $\langle a_1, b_2, c_0, d_0\rangle \rightarrow_{U^{\mathsf{asyn}}} \langle a_0, b_2, c_0, d_0\rangle$ tel que $\langle a_0, b_2, c_0, d_0\rangle \notin \mathsf{trace}(\mathbf{C})$.

Cette caractérisation qui permet de différencier entre un attracteur et un non-attracteur est importante pour la méthode de la recherche que nous proposons dans la suite. Pour un $n \in \mathbb{N}^*$ donné, nous énumérons de façon exhaustive tous les cycles de cette longueur n. Ensuite, puisque toute trace d'un cycle qui n'est pas un domaine piège, il n'est pas un attracteur (lemme 5.4 ci-dessus), alors nous développons un filtre qui est basé sur ce lemme et qui ne garde que les cycles dont les traces sont des domaines pièges. Ainsi, selon

le lemme 5.3 ci-dessus, toutes les traces des cycles pièges restants sont des attracteurs. Nous introduisons d'avantage cette méthode dans la sous-section 5.4.4 suivante.

Une propriété intéressante peut être observée sur les graphes d'états des figures 5.4 et 5.5 en pages 146-147 et qui relie les attracteurs trouvés selon les deux sémantiques de la dynamique des AN (l'asynchrone et le synchrone). En effet, le graphe d'états trouvé selon la sémantique asynchrone (figure 5.4 en page 146) partage son attracteur de taille 2 avec le graphe d'états en sémantique synchrone (figure 5.5 en page 147). Ainsi, dans le lemme 5.5 ci-dessous, nous montrons formellement ce résultat qui dit que tout attracteur de taille 2 calculé selon la sémantique asynchrone se trouve identiquement dans le graphe d'états calculé selon la sémantique synchrone.

Lemme 5.5 (Attracteurs de taille-2)**.** *Tout attracteur de taille 2 trouvé selon la sémantique asynchrone est aussi un attracteur selon la sémantique synchrone.*

Démonstration. Soit $\mathbf{A} = \{\zeta, \zeta'\}$ un attracteur de taille 2 trouvé selon la sémantique asynchrone. Donc, selon le lemme 5.3 en page 150, il existe un cycle qui parcourt tous les états globaux de $\mathbf{A}$. Soit $\mathbf{C} = (\zeta\,;\zeta'\,;\zeta)$ tel que $\zeta \rightarrow_{U^{\text{asyn}}} \zeta' \rightarrow_{U^{\text{asyn}}} \zeta$. De plus, puisque tout attracteur est un domaine piège (définition 5.12 en page 148), alors $\nexists \zeta'' \in \mathcal{S}$ tel que $\zeta \rightarrow_{U^{\text{asyn}}} \zeta'' \vee \zeta' \rightarrow_{U^{\text{asyn}}} \zeta''$. Sachant que la dynamique est calculée selon la sémantique asynchrone, donc toute transition globale jouable contient exactement une seule transition locale (définition 5.3 en page 136). Ainsi, nous pouvons en déduire qu'il n'existe qu'une seule transition locale qui est jouable dans chacun des états globaux de l'attracteur : $|\mathsf{J}(\zeta)| = 1$ et $|\mathsf{J}(\zeta')| = 1$.

Soient alors $\tau, \tau' \in \mathcal{T}$ telles que : $U^{\text{asyn}}(\zeta) = \{\{\tau\}\}$ et $U^{\text{asyn}}(\zeta') = \{\{\tau'\}\}$, avec $\mathsf{J}(\zeta) = \{\tau\}$ et $\mathsf{J}(\zeta') = \{\tau'\}$.

Maintenant, nous considérons le même modèle AN mais la dynamique est calculée selon une sémantique de mise à jour synchrone. Nous avons selon la définition 5.2 en page 135, l'ensemble des transitions globales jouables dans un état global ne dépend pas de la sémantique de mise à jour de la dynamique. Donc, même selon la sémantique synchrone $\mathsf{J}(\zeta) = \{\tau\}$ et $\mathsf{J}(\zeta') = \{\tau'\}$. Ainsi, selon la définition 5.4 en page 136, on a $U^{\text{syn}}(\zeta) = \{\{\tau\}\}$ et $U^{\text{syn}}(\zeta') = \{\{\tau'\}\}$. Par conséquent, le cycle $\mathbf{C} = (\zeta\,;\zeta'\,;\zeta)$ est aussi compatible avec la dynamique du AN calculée selon la sémantique synchrone : on a donc, $\mathsf{trace}(\mathbf{C}) = \{\zeta, \zeta'\} = \mathbf{A}$. Finalement, puisqu'aucune autre transition locale différente de τ (resp. de τ') n'est jouable dans ζ (resp. ζ'), alors, $\mathbf{A}$ est un domaine piège selon la sémantique synchrone. Ainsi, à partir du lemme 5.3 en page 150, on déduit que $\mathbf{A}$ est aussi un attracteur selon la sémantique synchrone. □

Bien que nous ne l'utilisions pas directement dans notre travail, cette caractéristique des attracteurs de taille 2 pourrait être utilisée pour accélérer le processus d'énumération des attracteurs. Cependant, il faut noter qu'elle ne s'applique qu'aux deux sémantiques étudiées dans la section 5.2 précédente (asynchrone et synchrone). En effet, dans le cas général, il n'est pas possible de déduire les attracteurs produits par un système selon une certaine sémantique en observant les attracteurs d'une autre.

Nous avons remarqué que dans un modèle AN qui est "booléen" (c'est-à-dire tous ses automates ont exactement 2 états locaux : 0 et 1), et en sémantique asynchrone, tous ses cycles, ainsi que tous ses attracteurs, sont de taille paire. En effet, chaque fois qu'on change l'état local d'un automate $a : a_0 \rightarrow a_1$, il faut activer exactement une transition

locale pour y revenir : $a_1 \to a_0$. Ce qui n'est pas le cas en multivalué : on peut imaginer par exemple, une telle évolution de a : $a_0 \to a_1 \to a_2$ et puis une transition qui fait retourner a dans son niveau initial $a_2 \to a_0$. Donc 2 transitions locales sont activées pour faire évoluer a de 0 vers 2 et une seule pour le faire revenir à l'état local initial (de 2 vers 0). Nous montrons ce résultat formellement dans le lemme 5.6 ci-dessous.

Lemme 5.6 (Cycles de longueur paire en asynchrone)**.** *Tout cycle en sémantique asynchrone et pour un modèle AN Booléen est de longueur paire.*

Démonstration. Dans un AN booléen, il n'y a que deux changements qui sont possibles dans un automate donné a : du niveau 0 au niveau 1 ($a_0 \to a_1$) ou du niveau 1 au niveau 0 ($a_1 \to a_0$). Puisqu'en sémantique asynchrone, il se produit exactement un changement entre deux états globaux successifs (définition 5.3 en page 136), donc à chaque fois qu'un automate a change l'état local initialement actif dans $\mathbf{C}_0[a]$, il faut exactement une transition pour y revenir afin d'avoir un cycle (i.e., avoir $\mathbf{C}_0 = \mathbf{C}_n$).

Ainsi, dans le cas général, dans un cycle de longueur $n \in \mathbb{N}^*$, s'il y a k changements d'automates qui se sont produits avec $k \in \mathbb{N}^*$ et tel que $\zeta_0[a] \to \overline{\zeta_0[a]}$ pour chaque automate a, alors il faut k changements inverses pour remettre tous ces automates à l'état local initial $\overline{\zeta_0[a]} \to \zeta_0[a]$, on a alors $n = 2k$. □

Exemple. Tous les cycles du graphe d'états trouvés selon la sémantique asynchrone illustrés dans la figure 5.4 en page 146 sont de longueurs paires. On donne quelques exemples :

- $\mathbf{C} = (\langle a_0, b_2, c_1, d_0\rangle ; \langle a_0, b_2, c_1, d_1\rangle ; \langle a_1, b_2, c_1, d_1\rangle ; \langle a_1, b_2, c_1, d_0\rangle ; \langle a_0, b_2, c_1, d_0\rangle)$ est de longueur $n = 4$ (tous ces états globaux sont colorés en jaune).
- $\mathbf{C}' = (\langle a_1, b_2, c_0, d_1\rangle ; \langle a_1, b_2, c_0, d_0\rangle ; \langle a_1, b_2, c_0, d_1\rangle)$ est de longueur $n = 2$ (son état global initial est coloré en vert).
- $\mathbf{C}'' = (\langle a_0, b_1, c_0, d_2\rangle ; \langle a_0, b_1, c_0, d_1\rangle ; \langle a_0, b_1, c_0, d_2\rangle)$ est de longueur $n = 2$ (tous ces états globaux sont colorés en bleu).

En asynchrone, l'activité de chaque transition globale correspond à celle de la transition locale qu'elle contient. Ainsi, une seule transition locale est activée dans chaque état global. Ce qui implique la parité de la longueur des cycles qui suivent une sémantique asynchrone de la dynamique. Par contre, en synchrone, ce n'est pas le cas, car plusieurs transitions locales sont activées simultanément dans une transition globale (toutes celles qui sont jouables). Par conséquent, la parité de la longueur des cycles en synchrone n'est pas garantie vu le nombre variable des transitions locales qui sont activées dans chaque état global.

Nous avons montré dans le lemme 5.3 en page 150 que tout attracteur (non singleton) est un cycle. Ainsi, on peut déduire que la taille de tout attracteur trouvé selon la sémantique asynchrone et dans un modèle AN booléen est aussi de longueur paire.

Corollaire 5.1. *Tout cycle* $\mathbf{C}$ *calculé selon la sémantique asynchrone de la dynamique et pour un modèle AN booléen, si* $\mathsf{trace}(\mathbf{C})$ *est un attracteur alors la longueur de* $\mathbf{C}$ *est nécessairement* paire.

Démonstration. Selon le lemme 5.3 en page 150, les attracteurs sont les traces des cycles pièges. Puisqu'en asynchrone et pour un modèle AN booléen, un cycle est toujours de longueur paire (voir lemme 5.6 ci-dessous), alors les cycles dont les traces sont des attracteurs sont aussi de longueur paire. □

Exemple. Prenons les cycles de l'exemple précédent, on trouve que les traces de **C** et de **C**″ sont des attracteurs :

- $\mathbf{A} = \mathsf{trace}(\mathbf{C}) = \{\langle a_0, b_2, c_1, d_0\rangle ; \langle a_0, b_2, c_1, d_1\rangle ; \langle a_1, b_2, c_1, d_1\rangle ; \langle a_1, b_2, c_1, d_0\rangle\}$.
- $\mathbf{A}'' = \mathsf{trace}(\mathbf{C}'') = \{\langle a_0, b_1, c_0, d_2\rangle ; \langle a_0, b_1, c_0, d_1\rangle\}$.

Ces 2 attracteurs cycliques **A** et **A"** sont les seuls qui sont obsevables dans le graphe d'états trouvés selon la sémantique asynchrone dans la figure 5.4 en page 146.

Ce résultat permet de réduire la recherche des attracteurs en sémantique asynchrone pour les modèles AN booléens. En effet, lors de l'exploration d'un réseau dans le but d'identifier ses attracteurs, il est suffisant de n'énumérer que ses cycle de taille paire, puis de vérifier si ces derniers sont des domaines pièges. Une seule exception dans laquelle on peut dire que la trace de tout cycle est un attracteur est dans le cas d'une sémantique de mise à jour de la dynamique synchrone purement déterministe. Ce cas se produit pour les modèles booléens, comme expliqué dans la section 5.2.3 en page 140.

Corollaire 5.2. *Tout cycle* **C** *calculé selon la sémantique synchrone de la dynamique pour un modèle AN Booléen, sa trace,* $\mathsf{trace}(\mathbf{C})$*, est un attracteur.*

Démonstration. Selon le lemme 5.3 en page 150, les attracteurs sont les traces des cycles pièges. Puisqu'en sémantique synchrone de la dynamique et pour un modèle AN booléen, chaque état global a exactement un seul état global comme successeur, alors, chaque successeur de tout état global visité par **C** appartient aussi à $\mathsf{trace}(\mathbf{C})$. Autrement dit, pour tous les états globaux de $\mathsf{trace}(\mathbf{C})$, il n'existe pas un successeur qui n'est pas dans $\mathsf{trace}(\mathbf{C})$. Ainsi, $\mathsf{trace}(\mathbf{C})$ est un domaine piège. □

D'autre part, dans le cas de la dynamique non-déterministe, l'identification des attracteurs nécessite une étude approfondie de la dynamique du AN. Et ceci est le cas des attracteurs complexes.

Attracteurs complexes

Nous considérons qu'un ensemble d'états globaux présente un *attracteur complexe*, quand la longueur minimale d'un chemin qui atteint tous ses états globaux est supérieure à la taille de cet attracteur. Autrement dit, si la taille d'un attracteur est égale à n, le chemin de longueur minimale qui visite tous les états globaux de l'attracteur est de longueur strictement supérieure à n.

C'est par exemple le cas de l'attracteur complexe représenté dans la figure 5.6 ci-dessous qui pourrait être appelé « attracteur en étoile ». Cette figure montre 4 états globaux notés par s_1, s_2, s_3 et s_4 et 6 transitions globales entre eux. Nous considérons qu'il s'agit d'un attracteur de taille 4 : $\mathbf{A} = \{s_1, s_2, s_3, s_4\}$.

Nous remarquons qu'il n'est pas possible de parcourir tous les éléments de cet attracteur par un chemin sans que ce dernier visite s_1 au moins 3 fois. Un exemple de chemin (qui est aussi un cycle) qui parcourt entièrement tous les états globaux de l'attracteur complexe est : $H = (s_1 ; s_2 ; s_3 ; s_2 ; s_4 ; s_2 ; s_1)$. H est de longueur égale à 6, et aucun chemin d'une longueur inférieure n'existe qui pourrait couvrir tous les éléments de cet attracteur bien que sa trace soit de taille 4.

Un *point fixe* (i.e., un état stable) est un cas spécial des attracteurs. En effet, il peut être assimilé à un attracteur de taille 1. Nous introduisons ce cas spécial des attracteurs dans la sous-section suivante.

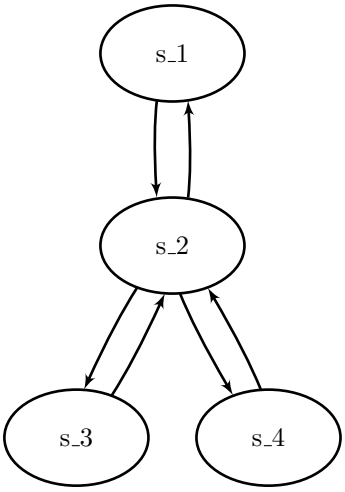


Figure 5.6 : Exemple simple mettant en valeur un attracteur complexe appelé aussi "attracteur en étoile".

5.4.3 Les attracteurs singletons (i.e., les points fixes)

Un point fixe est un attracteur de taille 1. Ainsi, il s'agit d'un état global sans successeur. Autrement dit, c'est tout état global dans lequel aucune transition globale n'est jouable (tel que c'est indiqué dans la définition 5.14 ci-dessous). Les états globaux de ce type ont un intérêt particulier car ils dénotent des situations dans lesquelles le modèle reste indéfiniment. L'existence de plusieurs points fixes dans un AN, dénote une multistabilité, et des bifurcations possibles dans la dynamique (Wuensche, 1998).

Définition 5.14 (Point fixe)**.** Soient $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$ un réseau d'automates et U une sémantique de la dynamique ($U \in \{U^{\text{asyn}}, U^{\text{syn}}\}$). Un état global $\zeta \in \mathcal{S}$ est appelé un *point fixe* (ou *état stable*) si et seulement si aucune transition globale n'est jouable dans ζ :

$$\mathsf{J}(\zeta) = \emptyset \Rightarrow U(\zeta) = \emptyset.$$

Il est à noter que l'ensemble des points fixes d'un modèle est le même dans les deux sémantiques de la dynamique : asynchrone et synchrone comme il est indiqué dans (Klarner, Bockmayr & Siebert, 2015) et dans (de Espanés, Osses & Rapaport, 2016). Nous formalisons dans le lemme 5.7 suivant cette propriété.

Lemme 5.7 (Les points fixes dans les sémantiques synchrone et asynchrone)**.** *Soit $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$ un réseau d'automates. L'ensemble des points fixes de $\mathcal{AN}$ est le même dans les dynamiques calculés selon les sémantiques asynchrone et synchrone :*

$$\forall \zeta \in \mathcal{S}, U^{\text{asyn}}(\zeta) = \emptyset \Longleftrightarrow U^{\text{syn}}(\zeta) = \emptyset.$$

Démonstration. Par définition, un point fixe est un état global dans lequel aucune transition globale n'est jouable. Autrement dit, aucune transition locale n'est jouable non plus. En effet, une transition globale n'est qu'un ensemble des transitions locales qui sont jouables (définition 5.5 en page 137). Par conséquent, si $\zeta \in \mathcal{S}$ est un point fixe selon la sémantique asynchrone (resp. synchrone), alors il n'existe aucune transition locale qui est jouable dans ζ. Puisque les conditions de la jouabilité d'une transition locale dans un état global donné sont les mêmes quelle que soit la sémantique de la dynamique (voir définition 5.2 en page

135), alors si une transition locale n'est pas jouable en asynchrone, elle n'est pas jouable en synchrone. Ainsi, il n'existe aussi aucune transition locale jouable dans ζ selon la sémantique synchrone (resp. asynchrone). □

Exemple. Bien que les figures 5.4 et 5.5 en pages 146-147 ne représentent pas toute la dynamique du modèle AN (car il existe d'autres états globaux qui ne sont pas présents), elles permettent de vérifier que dans les deux sémantiques, les points fixes sont les mêmes. En effet, elles représentent les mêmes trois points fixes colorés en rouge : $\langle a_1, b_1, c_1, d_0\rangle$, $\langle a_1, b_1, c_0, d_0\rangle$ et $\langle a_0, b_0, c_0, d_1\rangle$.

5.4.4 Idée intuitive des méthodes pour l'identification des attracteurs

Nous donnons dans cette sous-section, les pseudo-codes des algorithmes proposés pour l'identification des attracteurs dans un AN. Nous indiquons que nous donnons ces algorithmes uniquement afin de comprendre les idées derrière ces méthodes. En effet, ceci peut aider le lecteur qui n'est pas habitué avec le formalisme d'ASP (un paradigme logique) à comprendre ce que notre implémentation entend faire.

Attracteurs cycliques :

Nous décrivons ici comment obtenir les états des attracteurs qui peuvent être identifiés par des chemins dont la longueur minimale est fixée. L'obtention de tous les attracteurs de toutes les tailles peut être ainsi abordée en augmentant progressivement les longueurs des chemins considérées.

Le pseudo-code de cette méthode est donné par l'algorithme 4 ci-dessous. Il peut être résumé dans les quatre étapes suivantes :

1. énumérer tous les chemins de longueur n ;
2. éliminer tous les chemins qui ne sont pas des cycles (algorithme 5) ;
3. pour chaque trace d'un cycle restant, vérifier qu'il est aussi un domaine piège et donc c'est un attracteur (algorithme 6) ;
4. vérifier que n est la longueur minimale des chemins qui atteignent tous les états globaux de l'attracteur (algorithme 7).

Après les 3 premières étapes, les seuls chemins restants sont des cycles pièges. Ainsi, leurs traces sont nécessairement des attracteurs (voir lemme 5.3 en page 150). Nous ajoutons la 4$^{\text{ème}}$ étape pour optimiser le résultat de la méthode. En effet, nous exigeons le fait que chaque attracteur ne s'affiche que pour une seule valeur de n qui est égale à la longueur minimale du chemin qui atteigne tous ses états globaux. Autrement dit, quand la méthode est appelée avec plusieurs valeurs de n, elle ne retourne pas les mêmes attracteurs.

En effet, si l'attracteur est complexe, sa taille est surement inférieure à la longueur minimale du chemin qui peut atteindre tous ses éléments, n ; car il existe au moins un état global qui est visité plus qu'une fois par le cycle (voir exemple d'un attracteur complexe en page 156). Ainsi, la 4$^{\text{ème}}$ étape vérifie s'il s'agit d'un attracteur complexe, et si c'est le cas alors elle doit vérifier si le n choisi est égal à la longueur minimale pour que tous ses états globaux soient atteints.

D'autre part, s'il ne s'agit pas d'un attracteur complexe, la 4ème étape vérifie que la taille de l'attracteur doit être égale à n (voir propriété 5.1 en page 150). En effet, il peut s'agir d'un cycle piège d'une petite longueur mais qui recycle sur lui même ou qui donne à la fin un cycle de longueur supérieure à n. En revanche, ces plus petits cycles qu'il contient ne sont pas forcément indispensables pour trouver tous les états globaux de l'attracteur qu'il atteint (comme c'est le cas pour l'attracteur complexe). Ainsi, n n'est pas la longueur minimale avec laquelle cet attracteur doit être identifié. Par exemple, soit le cycle piège suivant, $(\zeta_0\,;\zeta_1\,;\zeta_0)$ dont la longueur est égale à $n=2$. L'attracteur qui est égal à sa trace est : $\{\zeta_0,\zeta_1\}$. Cet attracteur pourrait figurer aussi parmi les cycles de longueur $n=4$ s'il est répété deux fois comme suit : $(\zeta_0\,;\zeta_1\,;\zeta_0\,;\zeta_1\,;\zeta_0)$. Ainsi, bien que la trace de ce cycle piège soit un attracteur (qui n'est pas complexe), sa longueur ($n=4$) n'est pas minimale. Et donc, quand l'algorithme 4 ci-dessous est appelé avec $n=4$ en paramètre, cet attracteur $\{\zeta_0,\zeta_1\}$ ne sera pas affiché, mais il est seulement affiché quand $n=2$. Et ceci grâce à l'algorithme 7 de l'étape 4 qui garantit qu'il n'y aura pas l'identification des mêmes attracteurs pour des valeurs différentes de n.

Algorithm 4 : attracteurs$(\mathcal{AN}, U, n)$

Entrée :

- $\mathcal{AN}=(\Sigma,\mathcal{S},\mathcal{T})$: un AN
- $n\in\mathbb{N}^*\quad n\geq 2$
- $U\in\{U^{\text{asyn}},U^{\text{syn}}\}$: une sémantique de mise à jour de la dynamique du AN

Sortie :

- Δ_n : ensemble de tous les attracteurs tel que tous les états globaux de chaque attracteur sont parcourut par chemin de longueur minimale égale à n

Début :

$\Delta_n \longleftarrow \varnothing$ *// Initialisation*
pour chaque $\zeta\in\mathcal{S}$ **faire**
  $H=(H_i)_{i\in[\![0;n]\!]}\in\mathcal{S}^{n+1}$, avec $H_0=\zeta$ *// un chemin de longueur n est calculé // à partir de ζ selon la sémantique U*
  si (**verifCycle**$(H)=vrai$) **alors** *// voir algorithme 5*
    si (**verifDomPiege**$(\mathcal{AN},\text{trace}(H))=vrai$) **alors** *// voir algorithme 6*
      si (**verifMinChemin**$(H,n)=vrai$) **alors** *// voir algorithme 7*
        $\Delta_n \longleftarrow \Delta_n\cup\{\text{trace}(H)\}$ *// ajouter l'attracteur* trace(H) *dans* Δ_n
      fin si
    fin si
  fin si
fin pour chaque
retourner Δ_n

Algorithm 5 : verifCycle(H)

Entrée :

- $H = (H_i)_{i \in [\![0;n]\!]} \in \mathcal{S}^{n+1}$: un chemin

Sortie :

- $verif$: variable booléenne qui est égale à vrai si H est un cycle

Début

$verif \longleftarrow faux$ *// Initialisation*
si $(H_0 = H_n)$ **alors**
$verif \longleftarrow vrai$ *// le chemin H est un cycle*
fin si
retourner $verif$

Algorithm 6 : verifDomPiège$(\mathcal{AN}, \mathbf{E}, U)$

Entrée :

- $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$: un AN
- $\mathbf{E} \subset \mathcal{S}$: un ensemble d'états globaux
- U : une sémantique de mise à jour de la dynamique

Sortie :

- $verif$: variable booléenne qui est égale à vrai si **E** est un domaine piège dans $\mathcal{AN}$ selon la sémantique U

Début

$verif \longleftarrow Vrai$ *// initialisation*
pour chaque $\zeta \in$ **E faire** *// pour chaque état global dans* **E**
si $(\exists$ **Sce**$(\zeta) \notin$ **E**$)$ **alors** *// s'il existe un successeur qui n'est pas dans* **E**
$verif \longleftarrow faux$ *// si c'est le cas* **E** *n'est pas un domaine piège*
fin si
fin pour chaque
retourner $verif$.

Algorithm 7 : verifMinChemin(H, n)

Entrée :

- $H = (H_i)_{i \in [\![0;n]\!]} \in \mathcal{S}^{n+1}$: un chemin
- un entier $n \in \mathbb{N}^*$ $\quad n \geq 2$

Sortie :

- $verif$: variable booléenne qui est vrai si la trace de H ne peut être parcourut que par un chemin de longueur minimale égale à n

Début

si $(|\text{trace}(H)| = n - 1)$ **alors**
$verif \leftarrow vrai$ *// le chemin H est de longueur minimale égale à n*
sinon
si $(\exists\ H' = (H'_i)_{i \in [\![0;k]\!]} \in \mathcal{S}^{k+1}$ tel que $k < n \wedge \forall \zeta \in \text{trace}(H),\ \exists H'_i = \zeta \wedge verifCycle(H) = vrai \wedge$ si $j \neq i \Rightarrow H'_j \neq H'_i$ avec $0 < j < i < k)$ **alors**
$verif \leftarrow faux$ *// ∃ un autre chemin H′ de longueur k < n*
// tel qu'il parcourt tous les états globaux de trace(H)
sinon $verif \leftarrow vrai$
fin si
fin si
retourner $verif$

Points fixes :

Comme il a été indiqué précédemment, un point fixe n'est qu'un attracteur singleton. Ainsi, la méthode précédente qui identifie les attracteurs cycliques n'est plus applicable pour identifier un point fixe dans un AN. En effet, puisqu'un état singleton n'a pas de successeurs, alors il ne peut pas appartenir à une trace d'un cycle.

Par conséquent, nous séparons l'algorithme qui énumère les points fixes de celui qui énumère les attracteurs non-singletons cycliques. La méthode qui identifie tous les points fixes dans un AN peut être résolue avec un algorithme moins complexe. L'idée est simple, identifier tous les états globaux du réseau et puis vérifier pour chacun s'il a des transitons locales qui sont jouables. S'il n'existe aucune transition locale qui est jouable dans un état global ζ (c'est-à-dire $J(\zeta) = \varnothing$), alors ζ est un point fixe (voir algorithme 8 ci-dessous).

Il est clair que ces algorithmes montrent une complexité énorme. En revanche, grâce à l'utilisation d'ASP nous avons pu la contourner par la proposition des programmes logiques innovants. En effet, les résultats qui sont présentés dans le chapitre suivant montrent que nous pouvons identifier les attracteurs (singletons et cycliques) pour des modèles de taille assez large (jusqu'à 100 composants) au bout de quelques secondes ou quelques minutes. Par exemple, pour identifier les 5.875.504 points fixes d'un AN de taille égale à 100 automates, la méthode nécessite environ 2 minutes. Et pour identifier tous les attracteurs de taille $n = 2$ pour le même AN de taille 100 (en sémantique synchrone), la méthode renvoie 2.058.272 réponses au bout d'environ 5 minutes. Nous donnons plus de détails, dans le chapitre 6, sur les résultats obtenus pour des AN de différentes tailles modélisant des systèmes biologiques réels.

Algorithm 8 : pointsFixes($\mathcal{AN}$)

Entrée : $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$: un AN
Sortie : Δ : l'ensemble des points fixes de $\mathcal{AN}$
Début :

```
Δ ⟵ ∅                          // initialisation
pour tout ζ ∈ S faire
    si (J(ζ) = ∅) alors        // J(ζ) est l'ensemble des transitions locales
        Δ ⟵ Δ ∪ {ζ}            // jouables dans ζ
    fin si
fin pour tout
retourner Δ
```

5.5 Discussion

Le but de ce chapitre est d'étudier formellement la dynamique des RRB modélisés avec le formalisme des AN. Nous avons abordé principalement deux propriétés dynamiques : (i) la vérification de l'atteignabilité d'un objectif (qui est un ensemble d'états locaux des automates du réseau) à partir d'un état global donné ; et d'autre part (ii) l'identification des attracteurs singletons (i.e., points fixes) et non singletons (i.e., cycliques) dans un AN.

Nous notons que, puisque le formalisme des AN s'applique sur des modélisations de type modèle de René Thomas (Thomas, 1991), tous nos résultats sont également réalisables sur des modèles décrits avec le modèle de Thomas, ainsi que tout autre cadre qui peut être décrit en AN (tels que les réseaux booléens, les réseaux définis dans Biocham (Calzone, Fages & Soliman, 2006), etc.).

Le problème d'identification de tous les attracteurs dans un RRB est difficile. Même le plus simple problème de décider si le système a un point fixe, qui est considéré comme le plus petit attracteur est NP-complet. Certaines méthodes avec une moindre complexité consiste par exemple, à choisir de manière arbitraire un état global initial et à suivre un chemin suffisamment long, en espérant éventuellement trouver un attracteur. Mais ces méthodes ne sont pas exhaustives et peuvent manquer certains attracteurs (qui sont difficiles à atteindre).

Par conséquent, en l'absence de méthodes exhaustives efficaces, il est pertinent de développer une approche pour résoudre le problème NP-complet pour l'identification des attracteurs. D'où l'intérêt de notre approche présentée dans ce chapitre et qui consiste à examiner de manière exhaustive tous les états globaux possibles d'un réseau, ainsi que tous les chemins possibles de chacun de ces états. De toute évidence, cette méthode nécessite beaucoup de temps et de mémoire : 2^n états initiaux doivent être pris en compte pour un réseau booléen ayant n nœuds ; et les réseaux multi-valués augmentent encore plus cette valeur.

D'autre part, pour résoudre le problème de l'atteignabilité, un nombre suffisant de chemins doit être calculé pour s'assurer que toutes les trajectoires ont été explorées pour vérifier si l'objectif est effectivement atteignable ou pas à partir d'un état initial donné. De telles méthodes exhaustives nécessitent aussi énormément de temps et de mémoire.

Cette haute complexité pour l'étude de ces propriétés dynamiques justifie l'utilisation

d'un outil capable de traiter de tels problèmes difficiles. Donc, pour implémenter nos méthodes nous avons choisi le paradigme logique : Answer Set Programming. En effet, il s'est avéré efficace pour résoudre les problèmes complexes. Toutes les implémentations des méthodes présentées dans ce chapitre sont introduites dans le chapitre 6 suivant. Et nous y montrons aussi leur efficacité à résoudre ces problèmes NP-complets par leur applications sur des exemples de larges modèles biologiques (plus que 100 automates).

Chapitre 6

Mise en œuvre et résultats

Nous avons introduit dans les chapitres précédents de ce manuscrit, de nouvelles approches qui étudient la dynamique des réseaux de régulation biologiques (RRB). Rappelons que cette étude consiste : (i) à inférer les réseaux d'automates avec le temps (T-AN) modélisant un RRB à partir des données de séries temporelles (chapitre 4) ; (ii) et à analyser formellement les propriétés dynamiques des réseaux d'automates (vérification de l'atteignabilité et identification des attracteurs). Dans ce chapitre, nous présentons les implémentations de toutes ces méthodes en Answer Set Programming (ASP) et nous discutons les résultats obtenus par leurs applications sur des modèles biologiques. ASP a été identifié dans les années 1990 comme une sous-partie de la programmation logique, en tant que sémantique particulière, et est devenu l'un des domaines les plus dynamiques de la représentation des connaissances et la programmation déclarative. Il s'est avéré qu'ASP est particulièrement adapté à la résolution de problèmes combinatoires complexes. Nous montrons aussi dans notre étude que, grâce à l'utilisation d'ASP nous avons pu faire face aux problèmes combinatoires affrontés lors de l'étude des RRB.

Ce travail présente la partie de mise en oeuvre et application de tous nos articles publiés et qui sont cités dans les chapitres précédents : (Ben Abdallah et al., 2016) et sa version étendue dans (Ben Abdallah, Ribeiro, Magnin, Roux & Inoue, 2017) pour l'inférence des T-AN, (Ben Abdallah et al., 2015) pour la vérification de l'atteignabilité et (Ben Abdallah, Folschette, Roux & Magnin, 2017) pour l'identification des attracteurs.

6.1 Introduction

L'objectif principal de ce chapitre est de mettre en valeur les résultats théoriques développés durant cette thèse, et qui sont introduits dans les chapitres précédents de ce manuscrit. En effet, nous avons ainsi présenté dans le chapitre 3 le formalisme des réseaux d'automates

avec le temps (T-AN) et dans le chapitre 4 la méthode d'inférence des T-AN (modélisant un RRB) à partir des données de séries temporelles. Ensuite, dans le chapitre 5, nous avons introduit des méthodes d'analyse formelle des propriétés dynamiques des réseaux d'automates (AN) : (a) la vérification de l'atteignabilité d'un objectif à partir d'un état global initialement donné et (b) l'identification des attracteurs (ensemble d'états globaux à partir desquels la dynamique du réseau ne peut pas sortir) dans un AN.

Nous présentons ainsi les implémentations de toutes les méthodes qui sont introduites dans les chapitres 3, 4 et 5 et qui sont mises en œuvre principalement en programmation logique (Answer Set Programming, ASP). Ensuite, nous discutons des résultats obtenus par des expériences effectuées sur des modèles représentants des réseaux de régulations biologiques (RRB) réels.

Nous avons utilisé ASP (Baral, 2003) pour la plupart de nos implémentations. En effet, ASP est un paradigme déclaratif pour résoudre les problèmes qui apparaissent dans une démarche de la représentation des connaissances et le raisonnement. De plus, ASP est particulièrement adapté à la résolution de problèmes combinatoires complexes de recherche (Baral, 2008). Quelques exemples d'utilisation sont les applications de planification de la production, de configuration de produit, de diagnostic, et des problèmes de la théorie des graphes. Ainsi, nous introduisons dans la section 6.2 de ce chapitre, la syntaxe et la sémantique des règles logiques en ASP afin d'assurer une meilleure compréhension des parties de l'encodage données dans les sections qui suivent.

Ensuite, nous présentons dans la section 6.3 comment nous définissons un RRB modélisé avec le formalisme des T-AN en utilisant ASP comme langage de programmation. Et nous expliquons aussi dans cette section l'encodage de la méthode d'apprentissage des T-AN à partir des données de séries temporelles (appelée `MoT-AN`). Nous y détaillons aussi les parties de l'encodage en ASP des méthodes dédiées au raffinement des résultats obtenus par `MoT-AN`. Ce raffinement est effectué par des filtres qui ne permettent de garder que les modèles appris qui sont plus cohérents (i.e., les modèles appris ayant une dynamique et des propriétés dynamiques plus proches de celles du système que nous cherchons à modéliser).

Puis, nous développons dans la section 6.4, les programmes logiques en ASP qui permettent de vérifier la propriété d'atteignabilité. Cette propriété est détaillée dans la section 5.3 en page 142 du chapitre 5. Elle permet de trouver, s'il existe, un chemin dans la dynamique d'un AN qui assure l'activation d'un objectif à partir d'un état global initial. L'objectif peut être un état local d'un automate ou un ensemble d'états locaux de différents automates ou encore un état global du réseau.

Finalement, nous montrons dans la section 6.5 le scripts du code ASP de la méthode qui énumère de façon exhaustive les attracteurs d'un AN (comme il est indiqué dans la section 5.4 en page 146 du chapitre 5). Les attracteurs sont, en général, les ensembles d'états globaux dans lesquels la dynamique du système finit par atteindre et qu'elles n'en peut plus s'en échapper. Et si l'attracteur est un singleton, on l'appelle *"un point fixe"*.

Nous concluons ce chapitre avec une discussion, dans la section 6.6, à propos des résultats obtenus par les méthodes dont l'encodage est donné dans ce chapitre.

6.2 Programmation logique en Answer Set Programming

ASP a émergé à la fin des années 1990 comme un nouveau paradigme de programmation logique, ayant ses racines dans le raisonnement *non-monotone*, les bases de données déductives et la programmation logique avec négation par échec qui permet d'exprimer des exceptions, des restrictions et de représenter une connaissance incomplète. Une logique non-monotone est une logique formelle dans laquelle la base de faits inférés peut ne pas croître et même parfois décroître. En effet, la plupart des logiques formelles sont monotones, ce qui signifie qu'ajouter un fait ou un axiome à un ensemble de faits ou d'axiomes n'enlève pas de faits à cet ensemble. Autrement dit, cela signifie qu'ajouter une nouvelle connaissance à un système ne fera qu'augmenter les faits inférés dans ce système. Parce qu'elles interdisent la remise en cause de ce qui est su et déduit, les logiques monotones ne peuvent pas faire de la révision de connaissances.

Depuis sa création, ASP a été considéré comme une incarnation du calcul de raisonnement non-monotone et un outil efficace de représentation des connaissances. Ce point de vue est renforcé par l'émergence de solveurs très efficaces pour ASP. Il semble désormais difficile de contester que ASP a apporté une nouvelle vie à la programmation logique et du raisonnement non-monotone de recherche et est devenu une force motrice majeure pour ces deux domaines. ASP est un paradigme de programmation déclarative avec une sémantique connue comme la sémantique des ensembles de réponses (*answer sets*). Ce paradigme permet au programmeur de spécifier quels sont les résultats attendus et non pas comment les atteindre.

Dans le but de résoudre un problème, le programmeur conçoit un programme logique de sorte que les modèles du programme déterminent des solutions au problème. ASP a été identifié comme une sous-partie de la programmation logique, ayant une sémantique particulière, et est devenu l'un des domaines les plus dynamiques de la représentation des connaissances et de la programmation déclarative. Les programmes ASP sont composés d'un ensemble de faits et d'un ensemble de règles à partir desquelles d'autres faits peuvent être dérivés. Un ensemble de faits cohérents qui peut être dérivé à partir d'un programme à l'aide des règles est appelé *"ensemble de réponses"* pour le programme ASP. Les ensembles de réponses possibles pour un programme ASP sont calculés avec un outil appelé un *solveur*. Cette approche est étroitement liée à celle poursuivie dans le contrôle de satisfiabilité propositionnelle (SAT), où les problèmes sont codés comme des théories propositionnelles dont les modèles représentent les solutions au problème donné.

Nous introduisons successivement dans la suite, la syntaxe et la sémantique des programmes ASP (section 6.2.1), puis la démarche qui doit être suivie pour modéliser un programme ASP (section 6.2.2).

6.2.1 Syntaxe et la sémantique d'ASP

Dans cette section, nous récapitulons brièvement les éléments de base d'ASP (Baral, 2003). L'idée d'ASP est de représenter un problème de calcul, exprimé par un programme dont les ensembles de réponses (i.e., *answer sets*) correspondent à des solutions, et ensuite d'utiliser un solveur d'answer set pour trouver les solutions.

Les principaux avantages de l'ASP sont :

– sa simplicité,

– sa capacité à modéliser efficacement les spécifications incomplètes et des contraintes de fermeture (détaillée ci-dessous) qui permettent de résoudre un programme ASP, et
– son rapport à la satisfaction des contraintes et la satisfiabilité propositionnelle.

Un programme logique en ASP est un ensemble fini de *règles* de la forme suivante :

$$a_0 \leftarrow a_1, \ldots, a_m, \ not\ a_{m+1}, \ldots, \ not\ a_n. \tag{6.1}$$

où $n \geq m \geq 0$, tous les $a_1, \ldots, a_n$ sont des atomes propositionnels, et on note par le symbol "*not*" *négation par échec* et a_0 est un *atome propositionnel* ou $\perp$ (c'est-à-dire faux) est omis dans les règles d'ASP (nous en donnons plus de détails dans la suite, voir ligne 6.3 ci-dessous).

La lecture intuitive d'une telle règle 6.1 est que chaque fois que $a_1, \ldots, a_m$ sont connus pour être vrais et il n'y a aucune preuve pour que l'un des atomes niés $a_{m+1}, \ldots, a_n$ soit vrai, alors a_0 doit être vrai aussi.

Certaines règles spéciales sont remarquables. Une règle où $m = n = 0$ est appelée un "*fait*" et est utile pour représenter des données (ou des connaissances préalables) parce que l'atome gauche a_0 est toujours vrai. Il est souvent écrit sans la flèche centrale (voir ligne 6.2).

D'autre part, une règle où $n > 0$ et $a_0 = \perp$ est appelée une "*contrainte*" (voir règle 6.3). Comme $\perp$ ($a_0 = \perp$ et est remplacé par le vide dans la contrainte) ne peut jamais devenir vrai, si le côté droit d'une contrainte est vrai, cela invalide toute la solution. Les règles de type des contraintes sont donc très utiles pour filtrer les réponses non désirées. Le symbole $\perp$ est généralement enlevé dans une contrainte .

$$a_0. \tag{6.2}$$

$$\leftarrow a_1, \ldots, a_m, \ not\ a_{m+1}, \ldots, \ not\ a_n. \tag{6.3}$$

Dans le paradigme ASP, la recherche de solutions consiste à calculer les ensembles de réponses d'un programme donné. Un ensemble de réponses pour un programme est défini par Gelfond et Lifschitz (Gelfond & Lifschitz, 1988) comme suit. Une *interprétation* I est un ensemble fini d'atomes propositionnels. Une règle r donnée dans la ligne 6.1 est *vraie sous* I si et seulement si :

$$\{a_1, \ldots, a_m\} \subseteq I \wedge \{a_{m+1}, \ldots, a_n\} \cap I = \emptyset \Rightarrow a_0 \in I \ .$$

Une interprétation I est un *modèle* d'un programme P si chaque règle $r \in P$ est vraie sous I. Enfin, I est une *réponse* de P si I est un modèle minimal (en termes d'inclusion) de P^I, où P^I est défini comme le programme qui résulte de P en supprimant toutes les règles qui contiennent un atome nié qui apparaît dans I, et en supprimant tous les atomes niés des règles restantes.

Les programmes ASP peuvent ne produire aucune réponse, une réponse ou plusieurs réponses. Par exemple, le programme ASP suivant :

$$b \leftarrow not\ c. \tag{6.4}$$

$$c \leftarrow not\ b. \tag{6.5}$$

produit deux réponses : $\{b\}$ et $\{c\}$. En effet, l'absence de c rend b vrai, et inversement, l'absence de b rend c vrai.

Les "*cardinalités*" dans une règle ASP généralisent la création de plusieurs réponses. La façon la plus habituelle d'utiliser une cardinalité est à la place de a_0 dans la règle en ligne 6.1 ci-dessus comme c'est montré en ligne 6.6 ci-dessous :

$$l\ \{q_1,\ \ldots,\ q_k\}\ u\ \leftarrow\ a_1,\ \ldots,\ a_m,\ not\ a_{m+1},\ \ldots,\ not\ a_n. \tag{6.6}$$

où $k, l, u \in \mathbb{N}$ tels que $k \geq 1$, $l \geq 0$ et $u \geq 0$ et u peut être égal à l'infini (∞) quand il n'est pas spécifié.

Une telle cardinalité signifie que l'ensemble de réponses X doit contenir au moins l et au plus u atomes de l'ensemble des atomes $\{q_1, ..., q_m\}$ ou, en d'autres termes : $l \leq |\{q_1, ..., q_m\} \cap X| \leq u$ où $\cap$ est le symbole de l'intersection des ensembles et $|A|$ indique la cardinalité d'un ensemble A. Nous notons que cela peut créer plusieurs réponses dans l'ensemble de réponses car il peut y avoir de nombreuses solutions X à ce programme. Autrement dit, différents choix de combinaisons de $\{q_1, ..., q_m\}$ et qui satisfont le fait que leurs tailles soient comprises entre l et u. Ainsi, il y aura autant de réponses au programme que de combinaisons possibles. En utilisant les cardinalités, l'exemple du programme ASP ci-dessus, écrit dans les lignes 6.4 et 6.5, peut être résumé dans le programme ASP suivant et qui contient un seul fait :

$$1\ \{b, c\}\ 1. \tag{6.7}$$

S'ils ne sont pas explicitement donnés, l est par défaut égal à 0 et u est par défaut égal à ∞. De plus, si une telle cardinalité se trouve dans le corps d'une règle, elle ne crée pas plusieurs jeux de réponses. Par contre, la tête de la règle qui la contient n'est vrai que si la condition $l \leq |\{q_1, ..., q_m\} \cap X| \leq u$ est satisfaite dans l'ensemble de réponse.

Les atomes dans ASP sont exprimés sous forme de *prédicats* avec une arité, c'est-à-dire qu'ils peuvent s'appliquer à des *termes* (également appelés arguments). Par exemple, prenons le programme suivant, que nous suggérons pour mieux comprendre :

$$fishCannotFly. \tag{6.8}$$
$$fish(shark). \tag{6.9}$$
$$livesIn(X, water) \leftarrow fish(X),\ fishCannotFly. \tag{6.10}$$

La signification intuitive de ce programme (et plutôt de la règle 6.10) est que si les poissons ne volent pas (ce qui est le cas) et que quelque chose (i.e., X) est un poisson, alors cette chose vit dans l'eau.

Ici, $fishCannotFly$ est un prédicat avec une arité zero (sans termes), $fish(X)$ a une arité égale à un (un terme, définissant X en tant que poisson), et $livesIn(X, Y)$ a deux arités (tel que le premier terme X vit dans le deuxième terme Y). D'autre part, les termes $shark$ et $water$ sont des constantes tandis que X est une variable, qui peut représenter tout atome du programme.

Par convention, les noms des constantes dans un programme ASP commencent par une lettre minuscule ou bien ils sont écrits entre guillemets (e.g., "$water$" ou encore "$Water$"), et les noms des variables commencent par une lettre majuscule.

Cependant, résoudre un programme ASP, comme c'est expliqué ci-dessus, exige qu'il ne contienne aucune variable. Ainsi, une étape de *grounding* (c'est-à-dire stabilisation) par un *grounder* est d'abord nécessaire. En effet, elle consiste à supprimer toutes les variables en les remplaçant par des constantes tout en préservant la signification du programme. Dans l'exemple ci-dessus, le *grounder* produit le programme sans variables suivant, où X est remplacé par la seule constante $shark$:

$$fishCannotFly. \tag{6.11}$$
$$fish(shark). \tag{6.12}$$
$$livesIn(shark, water) \leftarrow fish(shark),\ fishesCannotFly. \tag{6.13}$$

Après la résolution par le *solveur*, le seul ensemble de réponses correspondant à ce programme est :

```
Answer 1: fishCannotFly  fish(shark)  livesIn(shark,water)
```

Pour les travaux de cette thèse, nous avons utilisé Clingo[1] (Gebser, Kaminski, Kaufmann, Ostrowski, Schaub & Wanko, 2016) qui est une combinaison des deux outils suivants : Gringo (le *grounder*) + Clasp (le *sloveur*).

6.2.2 Modélisation d'un programme ASP

On peut considérer que la modélisation d'un programme est l'une des composantes fondamentales de la démarche scientifique en informatique. Elle concerne tout système que nous cherchons à maîtriser et tout problème que nous cherchons à résoudre. Et pour le cas particulier d'un modèle d'un programme ASP, il possède deux caractéristiques principales :

- c'est une simplification d'un système donné et
- il permet de mettre au point une action qui pourrait être effectuée sur le système.

Le processus de modélisation peut être considéré comme une forme particulière d'opérationnalisation de la représentation de connaissances. Les programmes logiques d'ASP suivent la stratégie "générer et tester". Cette stratégie comprend quatre étapes :

- énumérer avec des faits,
- expliquer avec des règles,
- générer toutes les possibilités avec des cardinalités, et enfin
- filtrer avec des contraintes.

À titre d'exemple, considérons le problème bien connu des n-reines (ou *n-queens*). Le but est de placer n reines sur un échiquier de dimension n $\times n$ de sorte que deux reines n'apparaissent jamais sur la même ligne, colonne, ou diagonale. Pour simplifier ce problème on va le résoudre en suivant les étapes de modélisation d'un programme ASP énoncées ci-dessus.

[1] Nous avons utilisé Clingo version 5.0 : `http://potassco.sourceforge.net/`

Énumération avec des faits :

Pour commencer, il faut d'abord définir un échiquier par des faits dans un programme ASP. Étant donné qu'un échiquier n'est qu'un ensemble de cases dans lequel chaque case est caractérisée par ses coordonnées (I, J). Soient alors les atomes $row(I)$ et $col(J)$ qui indiquent respectivement le numéro de la ligne et le numéro de la colonne. Nous définissons ainsi dans la ligne 6.14 un échiquier de dimension $n \times n$ tel que n est une constante du programme ASP (dont la valeur est donnée comme entrée lors de l'appel du programme).

$$row(1..n).\ col(1..n). \tag{6.14}$$

Par conséquent, ces deux règles seront répandues et déclareront tous les atomes nécessaires pour définir toutes les lignes et toutes les colonnes d'un échiquier. Par exemple, si $n = 3$, on aura les faits suivants :

$$row(1).\ row(2).\ row(3).$$

$$col(1).\ col(2).\ col(3).$$

Expliquer avec des règles :

Soit $queen(I, J)$ l'atome qui positionne une reine dans la case de coordonnées (I, J) sur l'échiquier. Autrement dit, si $queen(I, J)$ est vrai dans un programme ASP, il existe une reine à la I$^{\text{ème}}$ ligne ($row(I)$) et J$^{\text{ème}}$ colonne de l'échiquier ($col(J)$) :

$$queen(I, J) \leftarrow row(I),\ col(J). \tag{6.15}$$

Générer toutes les possibilités avec des cardinalités :

Pour cet exemple de *n-queens*, la cardinalité doit créer tous les remplissages possibles de l'échiquier par les reines. Donc, chaque case de coordonnées (I, J) peut être remplie ou pas. Ainsi, nous ajoutons la cardinalité dans la règle de la ligne 6.15 ci-dessus et nous obtenons la règle de la ligne 6.16 ci-dessous. Cette dernière garde la même signification que la règle de la ligne 6.15 mais crée plusieurs réponses tel que chaque réponse présente un remplissage candidat de l'échiquier par des reines.

$$\{queen(I, J) : row(I),\ col(J)\}. \tag{6.16}$$

Filtrer avec des contraintes :

Dans cette étape de développement d'un programme ASP, nous proposons un ensemble de contraintes qui éliminent toute réponse candidate qui n'est pas une solution du problème considéré. Pour cet exemple, nous avons créé tous les remplissages de l'échiquier qui sont possibles à l'étape précédente et maintenant nous supprimons tout remplissage qui est incorrect.

La première contrainte à la ligne 6.17 (respectivement à la ligne 6.18) supprime tout remplissage dans lequel il y a plus qu'une reine sur la même colonne (respectivement sur la même ligne).

$$\leftarrow queen(I, J),\ queen(II, J),\ I != II. \tag{6.17}$$

$$\leftarrow queen(I, J),\ queen(I, JJ),\ J! = JJ. \tag{6.18}$$

De plus, les contraintes en lignes 6.19 et 6.20 ci-dessous élimine tout remplissage candidat dans lequel il existe plus qu'une reine sur les diagonales.

$$\leftarrow queen(I, J),\ queen(II, JJ),\ (I, J)! = (II, JJ),\ I - J == II - JJ. \tag{6.19}$$

$$\leftarrow queen(I, J),\ queen(II, JJ),\ (I, J)! = (II, JJ),\ I + J == II + JJ. \tag{6.20}$$

Et finalement, la contrainte 6.21 exige le fait que dans une solution il y a exactement n reines sur l'échiquier. En effet, comme c'est indiqué dans cette contrainte, la borne inférieure et la borne supérieure du nombre des $queen(I, J)$ dans une solution sont égales à n.

$$\leftarrow not\ n\{queen(I, J)\}n. \tag{6.21}$$

Dans le reste de ce chapitre, nous proposons des programmes ASP dont la sémantique et la syntaxe des règles se basent sur celles qui sont présentées ici. Nous commençons ainsi par présenter dans la section suivante la méthode pour apprendre les modèles T-AN à partir des données de séries temporelles et qui est introduite dans le chapitre 4.

6.3 L'inférence des T-AN

Les algorithmes d'inférence existants peuvent être une source d'inspiration pour de nouvelles méthodologies d'apprentissage de modèles. En particulier, ASP est une forme de programmation déclarative qui a été utilisée successivement dans de nombreuses représentations du savoir et de processus de raisonnement (Niemelä, 1999). En outre, il a été prouvé utile pour la reconstruction d'un réseau (Durzinsky, Marwan, Ostrowski, Schaub & Wagler, 2011) et l'inférence des réseaux métaboliques (Videla, Guziolowski, Eduati, Thiele, Gebser, Nicolas, Saez-Rodriguez, Schaub & Siegel, 2014).

Ainsi, nous avons profité de la puissance d'un tel langage de programmation pour implémenter notre méthode d'inférence. Nous rappelons que nous avons proposé dans le chapitre 4, une méthode d'apprentissage des RRB modélisés avec le formalisme des T-AN à partir des données de séries temporelles. Cette méthode est appelée `MoT-AN` (Modélisation des T-AN) et le pseudo-code de son algorithme est détaillé en page 93 du chapitre 4. Ainsi, nous présentons dans cette section l'implémentation de cet algorithme en ASP. Et nous indiquons que dans plusieurs endroits dans la suite nous nous référons à des définitions ou des figures du chapitre 4 afin d'assurer une meilleure compréhension du contenu.

6.3.1 La méthode de modélisation des T-AN, `MoT-AN`, en ASP

Comme mentionné précédemment dans la section 6.2.2, la première étape de la modélisation d'un programme ASP consiste à énumérer les données avec des faits. `MoT-AN` a 4 entrées que nous détaillons ci-dessous avec le cas d'étude du chapitre 4, repris dans les figures 6.1 et 6.2 ci-dessous :

- Un modèle T-AN dont l'ensemble des transitions locales temporisées est vide. Ainsi, il n'y a que des automates et leurs états locaux. Nous utilisons alors le prédicat `automatonLevel(X,Val)` où `X` est l'identifiant de l'automate et `Val` est son niveau

discret. À la ligne 8 du programme ASP ci-dessous, nous définissons les trois automates du réseau *a*, *b* et *z* avec leurs niveaux locaux 0 et 1.

- Les chronogrammes des données de séries temporelles. Chaque chronogramme présente une évolution discrète d'expression d'un automate au cours du temps. Nous les appelons aussi des *observations* et nous les codons par un ensemble de prédicats `obs(X,Val,T)`, où `X` est un identifiant d'automate, `Val` un niveau local de cet automate et `T` un point temporel tel que l'automate `X` est au niveau discret `Val` à l'instant `T` (voir lignes 10-24 ci-dessous qui décrivent les chronogammes de la figure 6.1 ci-dessous).

- Le graphe d'influences qui indique pour chaque composant du réseau quels sont les composants qui ont une influence régulatrice sur lui. Le prédicat `existInfluence(X,Y)` indique que l'automate `X` est régulé par l'automate `Y` (voir lignes 26-31 qui décrivent le graphe d'influences de la figure 6.2 ci-dessous).

- Une constante $n \in \mathbb{N}^*$ qui indique le *indegree* de chaque composant (i.e., le nombre maximal des régulateurs qui peuvent influencer simultanément un composant). Dans le formalisme des T-AN cet indegree peut se traduire par le nombre des automates inclus dans l'ensemble de condition de chaque transition locale temporisée. Dans le programme ASP que nous détaillons ici, et pour ne pas encombrer le contenu, nous n'introduisons que les transitions locales temposisées avec un indegree $n = 1$ (c'est-à-dire un seul régulateur par transition).

Exemple. Reprenons dans la figure 6.1 ci-dessous, les chronogrammes illustrant l'évolution des niveaux d'expression discrets des composants *a*, *b* et *z* du modèle T-AN de l'exemple jouet de la figure 4.7 en page 96. Les lignes 10–24 décrivent ces chronogrammes par des règles ASP. Et dans la figure 6.2 ci-dessous, on trouve le graphe d'influences entre les composants du modèle T-AN que nous cherchons à inférer. Ensuite, nous encodons ce graphe en ASP dans les lignes 26–31.

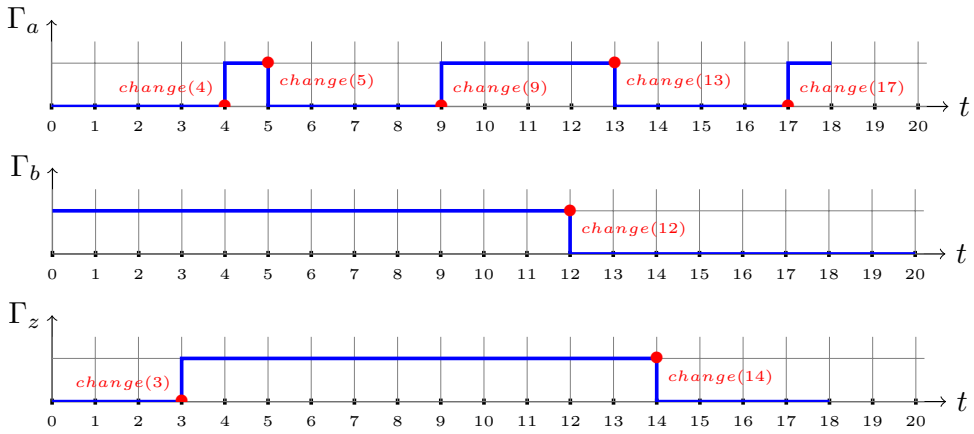


Figure 6.1 : Des chronogrammes trouvés par la simulation du modèle de l'exemple jouet (figure 4.7 en page 96) sur 18 unités de temps. Les chronogrammes Γ_a, Γ_b et Γ_z correspondent respectivement aux trois composants *a*, *b* et *z*. Chaque point rouge indique un changement dans la dynamique du système illustrée par un changement du niveau discret d'un composant à un instant donné. Par exemple, à l'instant $t = 4$, *change(4)* indique le changement de *a* du niveau 0 au niveau 1.

```
7 % Tous les composants du reseau avec leurs niveaux discrets apres la discretisation
8 automatonLevel("a",0..1). automatonLevel("b",0..1). automatonLevel("z",0..1).
```

```
9
10 % Donnees de series temporelles ou observation du composant "a"
11 obs("a",0,0). obs("a",0,1). obs("a",0,2). obs("a",0,3). obs("a",0,4). obs("a",1,4).
12 obs("a",1,5). obs("a",0,5). obs("a",0,6). obs("a",0,7). obs("a",0,8). obs("a",0,9).
13 obs("a",1,9). obs("a",1,10). obs("a",1,11). obs("a",1,12). obs("a",1,13). obs("a",0,13).
14 obs("a",0,14). obs("a",0,15). obs("a",0,16). obs("a",0,17). obs("a",1,17). obs("a",1,18).
15 % Donnees de series temporelles ou observation du composant "b"
16 obs("b",1,0). obs(""b",1,,1). obs("b",1,,2). obs("b",1,,3). obs("b",1,4). obs("b",1,5).
17 obs("b",1,6). obs("b",1,7). obs("b",1,8). obs("b",1,9). obs("b",1,10). obs("b",1,11).
18 obs("b",1,12). obs("b",0,13). obs("b",0,14). obs("b",0,15). obs("b",0,16). obs("b",0,17).
19 obs("b",0,18).
20 % Donnees de series temporelles ou observation du composant "z"
21 obs("z",0,0). obs("z",0,1). obs("z",0,2). obs("z",0,3). obs("z",1,3). obs("z",1,4).
22 obs("z",1,5). obs("z",1,6). obs("z",1,7). obs("z",1,8). obs("z",1,9). obs("z",1,10).
23 obs("z",1,11). obs("z",1,12). obs("z",1,13). obs("z",1,14). obs("z",0,14).
24 obs("z",0,15). obs("z",0,16). obs("z",0,17). obs("z",0,18).
```

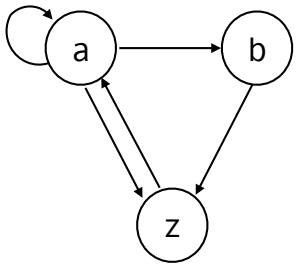


Figure 6.2 : Un graphe d'influences χ dont les nœuds sont : *a*, *b* et *z* et ils représentent les composants du RRB à modéliser. Chaque arc orienté représente une réaction régulatrice du composant régulateur (origine de l'arc) sur le composant régulé (destination de l'arc).

```
25 % Graphe d'influences
26 % Regulateurs de "a"
27 existInfluence("a","a"). existInfluence("a","z").
28 % Regulateurs de "b"
29 existInfluence("b","a").
30 % Regulateurs de "z"
31 existInfluence("z","a"). existInfluence("z","b").
```

La deuxième étape de modélisation d'un programme ASP consiste à "expliquer avec des règles". Nous présentons ainsi dans la suite, l'ensemble des règles qui définissent et résolvent le problème d'inférence des T-AN.

Nous définissons dans le prédicat `changeState(X,Val1,Val2,T)`, le point temporel `T` où l'automate `X` change son niveau de `Val1` vers `Val2`. Nous rappelons que nous admettons qu'à `t = 0`, tous les composants commencent à changer (lignes 34–35).

```
32 % Identification des changements ("changes")
33 % Initialisation : tous les composants commencent a changer a t=0 (hypothese)
34 changeState(X,Val,Val,0) ← obs(X,Val,0).
35 changeState(X,0) ← obs(X,_,0).
36 % Calculer les changements de chaque composant selon les observations (les chronogrammes)
37 changeState(X,Val1, Val2, T) ← obs(X, Val1, T), obs(X,Val2,T),obs(X, Val1, T-1),
38         obs(X, Val2, T+1), Val1!=Val2.
39 changeState(X,T) ← changeState(X,_,_,T).
```

Pour réduire la complexité du programme, nous considérons dans le prédicat `time` seulement les points temporels où les composants changent leur niveau (ligne 41). Il en va de même pour les délais : en effet, `D` est un délai qui est égal nécessairement à la différence entre deux points temporels auxquels certains composants changent leurs niveaux (ligne 42).

```
40 % Trouver tous les points de temps auxquels les changements se produisent
41 time(T) ← changeState(_,T).
42 delay(D) ← time(T1), time(T2), D=T2-T1, T2>=T1.
```

Ensuite, nous traitons les chronogrammes en les subdivisant en des sous-intervalles tels que dans chacun de ces intervalles chaque composant garde le même niveau local. En l'occurrence, pour un composant `a`, nous calculons les points temporels où il est constant (i.e., n'a qu'un seul niveau local) entre deux changements successifs quelconques (c'est-à-dire quel que soit des changements de `a` ou des autres composants) `T1` et `T2` :

```
43 % Traitement des observations
44 obs_normalized(X,Val,T1,T2) ← obs(X,Val,T1), obs(X,Val,T2), T1<T2, time(T1), time(T2),
45         not existChange(X,Val,T1,T2).
46 % Verifier si X change son niveau entre deux points temporels T1 et T2
47 existChange(X,Val,T1,T2) ← obs(X,Val,T1), obs(X,Val,T2), obs(X,Val1,T), T>T1, T<T2,
48         Val!=Val1.
49 existChange(X,Val,T1,T2) ← obs(X,Val,T1), obs(X,Val,T2), changeState(X,T), T>T1, T<T2.
50 existChange(X,Val,T1,T2) ← obs(X,Val,T1), obs(X,Val,T2), changeState(X,Val,Val1,T1),
51         T1<T2, Val!=Val1.
```

Dans notre apprentissage (de la méthode `MoT-AN`), le délai d'une transition locale temporisée est égal à la différence entre l'instant où le changement se produit (trouvé par le prédicat `changeState`) et l'instant correspondant au dernier changement des composants participants à la transition locale temporisée, qui sont : le composant régulé, `X`, et celui (ou ceux) qui le régule, `Y`. En plus, selon la sémantique de la dynamique des T-AN introduite dans le chapitre 3, il y a des composants qui peuvent agir "indirectement" sur l'activation d'une transition locale temporisée. Il s'agit des cas où il y a des conflits entre les transitions locales temporisées (quand elles partagent les mêmes ressources). Ainsi, il est possible de rencontrer dans la dynamique du T-AN une transition locale temporisée qui est jouable mais qui n'est pas activée à cause de son blocage par une autre transition qui est en conflit avec elle.

Ce calcul de délai δ est en fait effectué comme c'est montré dans l'extrait suivant de l'algorithme 1 en page 93, `MoT-AN`. Nous rappelons que **LS** est l'ensemble des états locaux des automates du T-AN, $n \in \mathbb{N}*$ est le nombre maximal des régulateurs par transition locale temporisée, $\mathcal{T}_{change(t)}$ l'ensemble des transitions locales temporisées apprises pour un changement produit à un instant t et Π_t^x est l'instant du dernier changement du composant x avant d'atteindre l'instant t.

- **pour chaque** $\ell \in \wp(N_a), |\ell| \leq n$ **faire**
  //pour chaque combinaison possible des régulateurs de a

$$\tau := a_i \xrightarrow[\delta]{\ell} a_j$$

telle que,

si $\exists\ \tau'{=}c_k \xrightarrow[\delta_c]{\ell_c} c_l \in \varphi$ avec $a_i \in \ell_c$, $\Pi_t^c \geq \Pi_t^a$, $\forall b_x \in \ell$, $\Pi_t^c \geq \Pi_t^b$ **alors**
//τ est bloquée par τ' donc τ n'est activée que quand τ' se termine à Π_t^c

$$\delta = t - \Pi_t^c$$

sinon *//τ est activée à partir de l'instant qui est égal au maximum des //instants entre les derniers changements de a et des automates de ℓ*

$$\delta = t - max(\Pi_t^a, \Pi_t^d)$$

avec $d_x \in \ell \wedge \forall b_x \in \ell,\ b \neq d,\ \Pi_t^b \leq \Pi_t^d$.

ajouter τ dans $\mathcal{T}_{change(t)}$.
//$\mathcal{T}_{change(t)}$ est l'ensemble des transitions apprises par changement à l'instant t

Ainsi, nous avons présenté cette partie de l'algorithme dans les règles ASP suivantes (lignes 56–58). Le prédicat `lastChange(X,Y,Max,T2)` calcule le dernier changement `Max` à partir duquel la transition locale temporisée recherchée et qui est responsable du changement de `X` a commencé son activité.

Si une transition locale temporisée qui change un composant `X` commence son activation à un instant `H`, ce dernier correspond au dernier changement de `X`, ou au dernier changement produit parmi les régulateurs de `X` (i.e., `Y`). On note que cet instant `H` peut aussi correspondre à l'instant auquel la ressource partagée `X` est débloquée par une autre transition locale temporisée qui est en conflit avec la transition recherchée.

```
52 % Trouver l'instant auquel la transition a commence son activation
53 % Le dernier changement Max entre X, Y et W tels que X est regule par Y et W est regule par X
54 last(X,Y,W,Max,T2) ← Max=#max{ T : changeState(Y,T;X,T;W,T) , T<T2}, changeState(X,T2),
55 existInfluence(X,Y), existInfluence(W,X), Max>=0.
56 % dernier changement que des regulateurs
57 lastRegul(X,Y,Max,T2) ← Max=#max{ T : changeState(Y,T;X,T) , T<T2}, changeState(X,T2),
58         existInfluence(X,Y), Max>=0.
59 % dernier changement s'il existe une transition en conflit avec celle cherchee
60 lastChangeConflit(X,Y,Max,T2) ← last(X,Y,W,Max,T2), transition(X,_,W,_,_,D, change(T3)),
61         lastRegul(X,Y,U,T2), T3<T2, T2-U>D.
62 % le dernier changement est le meme trouve dans lastChangeConflit
63 lastChange(X,Y,Max,T2) ← lastChangeConflit(X,Y,Max,T2).
64 % le dernier changement est le meme de lastRegul s'il n'existe pas de conflit
65 lastChange(X,Y,Max,T2) ← lastRegul(X,Y,Max,T2), not lastChangeConflit(X,Y,Max1,T2),
66         Max1!=Max, delay(Max1).
```

Pour créer autant de modèles que possible qui satisfont les données de séries temporelles données en entrée, nous ajoutons dans la tête de la règle les accolades `{}` entre les prédicats de la transition (lignes 69–71). Ceci correspond, en effet, à l'étape de génération de toutes les possibilités avec des cardinalités pour modéliser un programme ASP.

```
67 % Construire tous les modeles T-AN avec toutes les transitions locales temporisees candidates
68 % Generer toutes les transitions locales temporisees candidates
69 {transition(Y,Valy,X,Val1,Val2,D, change(T2))} ← obs_normalized(X,Val1,T1,T2),
70         obs_normalized(Y,Valy,T1,T2), changeState(X,Val1,Val2,T2), existInfluence(X,Y),
71         lastChange(X,Y,T1,T2), T2=T1+D, delay(D).
72 transition(Y,Valy,X,Val1,Val2,D) ← transition(Y,Valy,X,Val1,Val2,D, _).
```

Ensuite, nous nous assurons de la minimalité des modèles T-AN appris comme c'est indiqué à l'étape 2 de l'algorithme 1 :

$\forall t$ un point temporel dans Γ, si $\exists a \in \Sigma$ tel que a change de niveau à t alors,

$$|\mathcal{T}^{k}_{appris} \cap \mathcal{T}_{change(t)}| = 1.$$

Ceci consiste de conserver exactement une transition locale temporisée responsable d'un changement produit à un instant `T` et pour un composant `X`. En effet, plusieurs combinaisons sont possibles des régulateurs de `X` et qui sont candidates d'être la cause de ce changement (elles sont générées à l'étape 1 de l'algorithme 1).

Nous calculons donc dans la variable `Tot` du prédicat `getTransNumber(Tot,X,T)` (ligne 83) le nombre des différentes transitions apprises et qui existent dans un même modèle T-AN appris (c'est-à-dire dans le même ensemble de réponse). Puis, nous éliminons tous les modèles qui ne satisfont pas cette condition de minimalité par les contraintes de la ligne 85.

```
73 % pour chaque changement, garder une seule transition locale temporisee
74 % un xOR entre les modeles T-AN appris
75 % toutes les transitions possibles et qui sont apprises par changement
76 candidateTrans(Y,Valy,X,Val1,Val2,D, change(T2)) ← obs_normalized(X,Val1,T1,T2),
77         obs_normalized(Y,Valy,T1,T2), changeState(X,Val1,Val2,T2),
78         existInfluence(X,Y),lastChange(X,Y,T1,T2), T2=T1+D, delay(D).
79 % la transition selectionnee pour un T-AN appris
80 selected(Y,Valy,X,Val1,Val2,D,change(T2)) ← transition(Y,Valy,X,Val1,Val2,D),
81         candidateTrans(Y,Valy,X,Val1,Val2,D,change(T2)).
82 % le nombre des transitions selectionnees par changement
83 getTransNumber(Tot,X,T)← Tot={selected(_,_,X,_,_,_,change(T))}, changeState(X,T), T!=0.

84 % garder exactement une transition par changement dans le T-AN appris
85 ← getTransNumber(Tot,X,T), changeState(X,T), Tot!=1.
```

Nous rappelons que les T-AN appris doivent contenir des transitions locales temporisées qui sont cohérentes entre elles. C'est-à-dire s'il y a une transition dont le délai est calculé selon une autre qui est en conflit avec elle, alors ces deux transitions en conflit doivent appartenir au même T-AN appris. Ceci est satisfait directement dans le code que nous présentons et aucune contrainte n'est nécéssaire pour vérifier cette condition. En effet, dans les lignes 60–61, nous calculons le délai d'une transition s'il y a dans le même modèle, c'est-à-dire le même ensemble de réponse une transition en conflit avec elle. Et si cette transition n'est pas présente, donc le calcul sera fait normalement sans conflit avec la règle aux lignes 65–66 (d'où l'intérêt du `not` avant le prédicat `lastChangeConflit` dans cette règle).

Nous récapitulons dans la figure 6.3 ci-dessous une simplification pour expliquer le programme ASP présenté ci-dessus. En effet, elle illustre les informations qui sont nécessaires pour calculer l'origine, la destination, les conditions et le délai de chaque transition locale temporisée apprise.

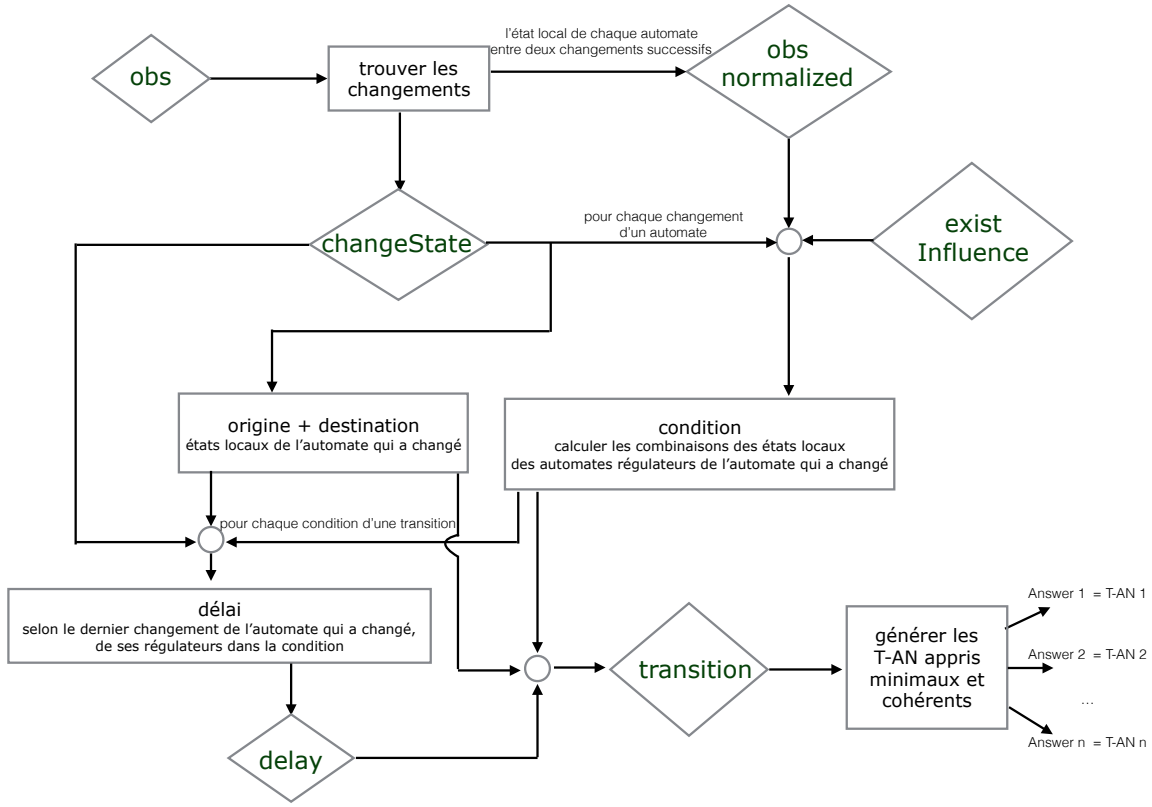


Figure 6.3 : Une illustration de la démarche suivie par le programme ASP pour calculer les transitions locales temporisées `"transition"` à partir des observations `"obs"` (i.e., les données de séries temporelles) et des influences entre les composants `existInfluence` (i.e., le graphe d'influences). Les noms dans les losanges correspondent aux prédicats dans le programme ASP.

Exemple. Nous donnons ici quelques exemples d'ensembles de réponses retournés par `MoT-AN` dont le programme ASP est décrit ci-dessous. Chaque ensemble représente un modèle T-AN appris à partir des chronogrammes et du graphe d'influences de l'exemple en pages 171-172 où nous donnons leurs descriptions en règles ASP.

Le programme retourne 48 ensembles de réponses, c'est-à-dire 48 T-AN sont appris. Ceci est cohérent avec ce qui est attendu et ce que nous l'avons montré dans la section 4.3.3 en page 99 du chapitre 4.

```
Answer: 1
transition("a",1,"a",1,0,1) transition("z",1,"a",0,1,4) transition("z",0,"a",0,1,3)
transition("a",1,"b",1,0,3) transition("a",0,"z",1,0,1) transition("a",0,"a",0,1,1)
transition("b",1,"a",0,"z",0,1,3)
Answer: 2
transition("z",1,"a",1,0,1) transition("z",1,"a",0,1,4) transition("z",0,"a",0,1,3)
transition("a",1,"b",1,0,3) transition("a",0,"z",0,1,3) transition("a",0,"a",0,1,1)
transition("a",0,"b",0,"z",1,0,1)
Answer: 3
transition("a",0,"b",1,"z",0,1,3) transition("a",0,"a",0,1,4) transition("z",1,"a",1,0,1)
transition("a",1,"b",1,0,3) transition("b",0,"z",1,0,2)
Answer: 4
transition("a",1,"a",1,0,1) transition("z",1,"a",0,1,4) transition("z",0,"a",0,1,3)
transition("a",1,"b",1,0,3) transition("a",0,"z",0,1,3) transition("a",0,"a",0,1,1)
transition("b",0,"a",0,"z",1,0,1)
...
```

6.3.2 Les filtres du raffinement du résultat de MoT-AN en ASP

Dans cette section, nous présentons la partie de l'encodage en ASP des filtres présentés dans la section 4.4 du chapitre 4. Nous rappelons que ces filtres optimisent le résultat des T-AN appris par MoT-AN. Ainsi, les modèles appris sont plus cohérents avec la sémantique de la dynamique des T-AN appris. En plus, l'étape de "filtrer avec des contraintes" est la dernière dans la démarche pour la modélisation d'un programme ASP (comme c'est indiqué dans la section 6.2.2 en page 168).

6.3.2.1 Filtre de cohérence entre la dynamique du modèle et la dynamique des T-AN

Selon la définition 4.4 en page 111, (qui introduit le filtre F1) toute transition locale temporisée qui est jouable à un instant donné et qui n'est pas en conflit avec une autre qui est en cours (i.e., franchissable), doit être activée (comportement synchrone). Autrement dit, si une transition locale temporisée τ est jouable à l'instant t tel que $\mathsf{ori}(\tau) = a_i$, $\mathsf{dest}(\tau) = a_j$ et $\mathsf{delai}(\tau) = \delta$, alors à l'instant $t + \delta$, l'automate a change de l'état local a_i vers a_j. Et si ce n'est pas le cas, alors le modèle T-AN appris est considéré comme non correct car il n'est pas cohérent avec la sémantique de la dynamique des T-AN. Ainsi, il sera éliminé par la contrainte suivante dans les lignes 107–110.

```
87 % Filtre F1
88 % Le dernier instant dans les donnees de series temporelles (\ie taille du chronogramme)
89 timeSeriesSize(Max) ← Max=#max{ T : obs(_,_,T) }.
90 step(0..Max)← timeSeriesSize(Max).
91 % Le niveau discret du composant X entre T1 et T2 ne change pas
92 obsT(X,Val,T1,T2) ← obs(X,Val,T1), obs(X,Val,T2), T1<T2, not existsChange(X,Val,T1,T2).
93 existsChange(X,Val,T1,T2) ← obs(X,Val1,T), T>T1, T<T2, Val1!=Val, obs(X,Val,T1), obs(X,Val,T2).
94
95 % Il y a une transition en conflit avec "transition(Y,Valy,X,Val1,Val2,D1)" entre T1 et T2
96 % (car elle partage la meme ressource X) et ainsi, s'il n'y a pas de changement de X
97 % c'est a cause de cette transition qui est en conflit avec elle
98 existTransInConflict(Y,Valy,X,Val1,Val2,D1,T1,T2) ← transition(Y,Valy,X,Val1,Val2,D1),
99         D1=T2-T1, T1<T2, obsT(X,Val1,T1,T2), obsT(Y,Valy,T1,T2), T3>=T2, T3-D2 <=T2,
100        transition(X,Val1,_,_,_,D2,change(T3)), D2>=D1,  step(T1), step(T2) .
101
102 % Eliminer tout modele dont la dynamique qui ne respecte pas la dynamique des T-AN
103 % Autrement dit, il existe une transition qui est franchissable (\cad elle est jouable
104 % et elle n'est pas en conflit avec aucune autre transition qui est en cours) mais elle
105 % n'est pas activee (\cad le changement de l'automate n'a pas eu lieu)
106 % voir definition 3.8 en page 68
107 ← transition(Y,Valy,X,Val1,Val2,D), obsT(X,Val1,T1,T2), obsT(Y,Valy,T1,T2),
108        not changeState(X,Val1,Val2,T2), D=T2-T1, step(D),
109        not existTransInConflict(Y,Valy,X,Val1,Val2,D,T1,T2),
110        T2!=Max, timeSeriesSize(Max).
```

6.3.2.2 Filtre basé sur la fréquence d'apparition des transitions locales temporisées dans les T-AN appris

Comme indiqué dans la définition 4.5 en page 113, en appliquant ce filtre F2 nous ne voulons garder que les transitions locales temporisées ayant une fréquence d'apparition élevée dans les modèles T-AN appris. Rappelons que ce filtre se base sur le fait que chaque modèle T-AN appris a au moins une simulation possible qui peut reproduire les chronogrammes

(c'est-à-dire les données de séries temporelles) pris en entrée. Ainsi, si une transition locale temporisée apparait dans plusieurs modèles, alors elle a une probabilité d'apparition élevée.

En ASP, l'option "- -*cautious*" calcule ce qu'ils appellent les *conséquences prudentes* : c'est le résultat de l'intersection de tous les ensembles de réponses d'un programme ASP. En effet, l'option "- -*cautious*" affiche des nouveaux ensembles de réponses dans lesquels on trouve les prédicats qui apparaissent plus fréquemment dans les ensembles de réponses initiaux (c'est-à-dire sans l'appel de - -*cautious*). Autrement dit, si par exemple, il y a un prédicat *a* qui n'apparait que dans un seul ensemble de réponses parmi les *n* ensembles de réponses (c'est-à-dire les solutions) du programme ASP, alors *a* ne va plus s'afficher dans aucun ensemble de réponse après l'application de l'option - -*cautious*. En revanche, s'il y a un autre prédicat *b* qui apparait dans les *n* ensembles de réponses, alors *b* fera toujours partie des ensembles de réponses retournés après l'application de l'option - -*cautious*.

Ainsi, c'est exactement ce que nous cherchons à afficher : ne garder que les prédicats `transition` qui apparaissent plus fréquemment dans les modèles T-AN appris retournés dans les ensembles de réponses du programme ASP de la méthode d'apprentissage `MoT-AN`. Par conséquent, nous utilisons cette option lors de l'appel du programme ASP détaillé ci-dessus pour l'apprentissage des modèles T-AN. Nous rappelons que les conséquences de ce filtre sur le résultat de l'apprentissage sont détaillées dans la section 4.4.2 en page 116 du chapitre 4. En effet, il réussit à optimiser le résultat de `MoT-AN` en enlevant un nombre important des transitions locales temporisées qui sont apprises en trop.

6.3.2.3 Filtre associé au déterminisme des régulateurs entre composants

La contrainte ci-dessous en ligne 114 garantit que chaque modèle appris respecte le filtre F3 introduit dans la définition 4.6 en page 115. Rappelons que ce filtre exige le fait que dans un modèle T-AN, chaque composant ne peut pas inhiber et activer un autre dans le même état local.

```
111 % Filtre F3 (1er cas)
112 % Y avec le meme niveau discret Valy ne peut pas activer (Val1<Val2) et inhiber (Val3>Val4)
113 % le meme composant X
114 ← transition(Y,Valy,X,Val1,Val2,_), transition(Y,Valy,X,Val3,Val4,_), Val1<Val2, Val3>Val4.
```

De plus, ce même filtre de la définition 4.6 en page 115, exige qu'un composant ne peut pas avoir le même effet sur un autre avec un même niveau discret. Par exemple, si l'automate *a* est actif (c'est-à-dire à l'état local a_1) active le composant *b*, alors quand *a* est inactif (c'est-à-dire à l'état local a_0), il ne peut plus activer *b*. Cette propriété est encodée en ASP par la contrainte à la ligne 118.

```
115 % Filtre F3 (2eme cas)
116 % Y avec differents niveaux discrets Valy1 et Valy2 regule un autre composant X
117 % avec le meme signe de regulation (de Val1 vers Val2)
118 ← transition(Y,Valy1,X,Val1,Val2,_), transition(Y,Valy2,X,Val1,Val2,_), Valy1!=Valy2.
```

6.3.2.4 Filtre associé à un délai moyen d'une transition locale temporisée

Nous présentons dans cette sous-section, un ensemble de raffinements qui peuvent être effectués sur les délais des transitions locales temporisées. En effet, l'apprentissage des T-AN est fait à partir des données de séries temporelles de systèmes biologiques réels qui

sont en général des données bruitées. Ainsi, les modèles appris peuvent contenir des modèles T-AN appris dans lesquels il y a des transitions locales temporisées qui sont identiques (c'est-à-dire elles ont la même origine, la même destination et la même condition) mais avec des délais différents. Par conséquent, nous suggérons plusieurs types de raffinement de ce cas de raffinement : (i) fusionner toutes les transitions locales temporisées identiques en une seule dont le délai est égal à l'intervalle qui englobe tous les délais trouvés (filtre F4 en définition 4.7 en page 116) ; (ii) ou bien le délai de cette transition après la fusion est égal à la valeur moyenne de tous les délais (filtre F4 révisé en définition 4.8 en page 116) ; (iii) ou exiger le fait que dans chaque modèle T-AN appris, chaque transition caractérisée par son origine, sa destination et sa condition a un seul délai (filtre F4 consolidé définition 4.9 en page 117). L'encodage en ASP de chacun de ces filtres est détaillé ci-dessous.

(i) Intervalle des délais :

Soit l'intervalle des délais `[Max, Min]` ; avec `Max` (resp. `Min`) est la valeur maximale (resp. minimale) des délais d'une transition locale temporisée donnée parmi toutes celles qui lui sont identiques dans le même T-AN appris. Les valeurs de `Max` et de `Min` sont calculées par les prédicats respectifs `maxDelay` (lignes 122–123), et `minDelay` (lignes 124–125). Enfin, la transition locale temporisée trouvée après la fusion est définie par le prédicat `transIntervalDelay` dans les lignes 126–127.

```
119 % Filtre F4
120 % Calculez la valeur maximale et la valeur minimale des delais
121 % des transitions locales temporisee identiques
122 maxDelay(Y,Valy,X,Val1,Val2,Max) ← Max=#max{ D : transition(Y,Valy,X,Val1,Val2,D)},
123         transition(Y,Valy,X,Val1,Val2,_).
124 minDelay(Y,Valy,X,Val1,Val2,Min) ← Min=#min{ D : transition(Y,Valy,X,Val1,Val2,D)},
125         transition(Y,Valy,X,Val1,Val2,_).
126 transIntervalDelay(Y,Valy,X,Val1,Val2,interval(Min,Max)) ← minDelay(Y,Valy,X,Val1,Val2,Min),

127         maxDelay(Y,Valy,X,Val1,Val2,Max).
```

(ii) Délai moyen :

Autrement, nous pouvons fusionner toutes les transitions locales temporisées qui ne se diffèrent que par leur délai dans une seule transition locale temporisée dont le délai est égal à leur délai moyen. Pour calculer ce délai moyen, nous procédons comme c'est indiqué dans la définition 4.8 en page 116, mais en ASP.

Nous calculons alors le nombre total de ces transitions locales temporisées identiques dans la variable `Tot` par le prédicat `nbreTotTrans(Y,Valy,X,Val1,Val2,Tot)`, où la partie commune entre toutes les transitions locales temporisées identiques est `Y, Valy, X, Val1, Val2`. Ensuite, la somme de tous ces délais est calculée dans la variable `S` par le prédicat `sumDelays` (lignes 134–135). Ainsi, pour trouver la valeur moyenne des délais, nous divisons `S` par `Tot` (`Davg = S / Tot`). La nouvelle transition locale temporisée trouvée après la fusion est alors `transAvgDelay(Y,Valy,X,Val1,Val2,Davg)` (lignes 137–138).

```
128 % Filtre F4 revise
129 % Transitions locales temporisees identiques avec un delai moyen
130 % Nombre des transitions identiques
131 nbreTotTrans(Y,Valy,X,Val1,Val2,Tot) ← Tot={transition(Y,Valy,X,Val1,Val2,_,_)},
132         transition(Y,Valy,X,Val1,Val2,_).
133 % La somme des delais
```

```
134 sumDelays(Y,Valy,X,Val1,Val2,S) ← S=#sum{ D: transition(Y,Valy,X,Val1,Val2,D)},
135         transition(Y,Valy,X,Val1,Val2,_), S!=0.
136 % La valeur moyenne des delais
137 transAvgDelay(Y,Valy,X,Val1,Val2,Davg) ← nbreTotTrans(Y,Valy,X,Val1,Val2,Tot),
138         sumDelays(Y,Valy,X,Val1,Val2,S), Davg=S/Tot.
```

(iii) Délai déterministe :

Selon la définition 4.9 en page 117, une transition locale temporisée n'a qu'un seul délai dans un modèle T-AN. Ainsi, quand la méthode d'apprentissage génère tous les modèles T-AN appris, nous ajoutons une contrainte en ligne 141 qui élimine tout T-AN qui ne satisfait pas cette propriété.

```
139 % Filtre F4 consolide
140 % Pas de delais differents pour la meme transition locale temporisee
141 ← transition(Y,Valy,X,Val1,Val2,D1), transition(Y,Valy,X,Val1,Val2,D2), D1!=D2.
```

Nous notons que tous les résultats montrés et discutés au chapitre 4 sont réalisées par les programmes ASP détaillés ci-dessus. Notamment les résultats après l'application des filtres sur l'exemple jouet dans la section 4.4 en page 110.

6.3.3 Application

Dans cette section, nous fournissons deux évaluations de l'algorithme 1, `MoT-AN`, implémentée en ASP[2] (comme c'est indiqué dans la section précédente). Nous évaluons la capacité de notre algorithme pour apprendre des modèles T-AN à partir des données de séries temporelles issues des RRB réels. En plus, nous évaluons l'impact de la quantité des données de séries temporelles sur le temps d'exécution. Ici, nous traitons les données de séries temporelles obtenues des compétitions internationales "*DREAM Challenges*"[3]. Les challenges de DREAM sont des défis récurrents d'ingénierie inverse qui fournissent des études de cas biologiques. Dans cette évaluation nous nous concentrons sur les ensembles de données provenant de DREAM4 challenge (Prill, Saez-Rodriguez, Alexopoulos, Sorger & Stolovitzky, 2011) et de DREAM8 challenge (Hill, Heiser, Cokelaer, Unger, Nesser, Carlin, Zhang, Sokolov, Paull, Wong et al., 2016).

6.3.3.1 DREAM4

Dans cette section, nous évaluons l'efficacité de notre algorithme 1, `MoT-AN`, à travers des études de cas provenant du DREAM4 challenge[4]. Les données fournies sont pour des systèmes de tailles différentes (10 gènes d'une part et 100 gènes de l'autre), ce qui nous permet d'évaluer le passage à l'échelle de notre approche. Nous décrivons dans la suite ces données et les résultats obtenus.

[2] Tous les programmes, décrits dans ce chapitre pour l'apprentissage de réseaux d'automates avec le temps sont disponibles sur : `http://www.irccyn.ec-nantes.fr/~benabdal/modeling-biological-regulatory-networks.zip`

[3] `http://dreamchallenges.org/`

[4] disponible en ligne à l'adresse suivante : `https://www.synapse.org/#!Synapse:syn3049712/wiki/74630`

Présentation des données :

Les données d'entrée que nous abordons ici sont les suivantes : 5 systèmes différents composés chacun de 100 gènes, tous issus du système de *Escherichia coli*. Pour chacun de ces systèmes, les données disponibles sont les suivantes : (i) 10 séries temporelles avec 21 points temporels, telles que 1000 est la taille de chaque série temporelle (i.e., la taille des chronogrammes) ; (ii) point fixe pour le système ayant le type sauvage (i.e., sans perturbations) ; (iii) des points fixes après le *knockout* de chaque gène (c'est-à-dire supprimer l'expression de ce gène) ; (iv) des points fixes après le *knock-down* de chaque gène (c'est-à-dire forcer la taux de transcription de ce gène à 50 %) ; (v) points fixes après quelques perturbations multifactorielles aléatoires. Nous avons traité dans nos expériences toutes ces données. Ainsi, pour appliquer notre algorithme, nous avons manipulé les données de séries temporelles décrites en (i).

Chaque série temporelle comprend des perturbations différentes qui sont maintenues tout le temps pendant les 10 premiers points temporels et appliquées à au plus 30 % des gènes. Dans ce contexte, une perturbation signifie une augmentation ou une diminution significative des valeurs d'expression génique. Dans les données brutes de la série temporelle, les valeurs d'expression génique sont données sous la forme de nombres réels compris entre 0 et 1.

La discrétisation est un élément crucial pour l'abstraction des données. Cependant, dans cette section, nous n'introduisons pas l'encodage de la méthode de la discrétisation. En fait, la représentation théorique de la méthode de la discrétisation est donnée dans la sous-section 4.2.1.1 en page 86. En utilisant des bases de données biologiques et des méthodes statistiques, nous pouvons identifier les seuils d'expression génique. Les techniques de regroupement pourraient être utilisées pour grouper des points de données en niveaux discrets. En revanche, plutôt que d'envisager des seuils, nous pourrions nous concentrer sur l'évolution de la vitesse des composants pour discrétiser les données. C'est en effet une perspective d'optimisation pour régulariser le modèle à travers différents niveaux et méthodes de discrétisation.

Cependant, dans cette section, nous nous concentrons sur l'algorithme d'apprentissage et considérons la discrétisation comme plus ou moins donnée. L'évaluation est faite par le calcul du *score* appelé le MSE (Mean Square Error). Ce score est le rapport de la différence entre le résultat attendu par le challenge et le résultat trouvé par le modèle appris par `MoT-AN`.

Résultats :

Pour appliquer notre approche, nous avons choisi de discrétiser ces données en 2 à 6 valeurs qualitatives (i.e., 6 seuils de discrétisation). Les résultats (détaillés ci-dessous) ont montré qu'augmenter le nombre de valeurs qualitatives de 2 à 4 améliore la précision, mais le score diminue de 5. Ceci est probablement dû à l'ajustement excessif : les relations apprises deviennent trop précises (c'est-à-dire les transitions locales temporisées pour le formalisme des T-AN) au point qu'elles ne peuvent pas être appliquées à autre chose que les données d'entraînement (c'est-à-dire les données à partir desquelles l'apprentissage est effectué). Le meilleur score obtenu avec 4 valeurs qualitatives est indiqué dans le tableau 6.1 ci-dessous. Chaque gène est discrétisé d'une manière indépendante, par rapport à la procédure suivante : nous calculons la valeur moyenne de l'expression du gène parmi toutes les données de

sa série temporelle, alors les valeurs entre la moyenne et la valeur maximale / minimale sont divisées en autant de niveaux. La discrétisation des données en fonction de la valeur moyenne de l'expression devrait réduire l'impact de la perturbation sur la discrétisation et donc sur le modèle appris.

Benchmark	Nombre des gènes	MSE	Benchmark	Nombre des gènes	MSE
1	10	0.086	6	100	0.052
2	10	0.080	7	100	0.042
3	10	0.076	8	100	0.033
4	10	0.039	9	100	0.033
5	10	0.076	10	100	0.052

Table 6.1 : Évaluation de notre méthode `MoT-AN` sur l'apprentissage et la prédiction de l'évolution des benchmarks à partir du DREAM4 challenge à travers le score qui est l'erreur quadratique moyenne (MSE, Mean Square Error). Les benchmarks sont composés de 10 gènes à gauche, et 100 gènes à droite.

Le DREAM4 challenge offre deux problèmes différents, qui consistent à prédire : (i) la structure des interactions géniques (sous forme d'un graphe orienté non signé) et (ii) les attracteurs dans certaines conditions données. Notre méthode n'est pas conçue pour aborder la première question. En effet, nous devons connaître le graphe d'influences pour exécuter l'algorithme `MoT-AN`. Cependant, les modèles que nous apprenons peuvent être appliqués pour prédire les chemins et donc les attracteurs. Ici, nous considérons les graphes d'influences trouvés dans le premier problème (i.e., dans (i)) comme une connaissance donnée et qui est inférée par l'outil Gene Network Weaver (Schaffter, Marbach & Floreano, 2011). Ainsi, nous abordons le problème de la prédiction des attracteurs (i.e., dans (ii)) pour évaluer nos résultats.

Pour cette évaluation, le DREAM4 challenge fournit 1 état initial et 5 différentes conditions de double *knockout* des gènes (c'est-à-dire supprimer 2 gènes à la fois). Le but est de prédire l'attracteur dans lequel le système tombera de l'état initial pour chaque double knockout. Ici, nous choisissons simplement le premier modèle T-AN que notre méthode ASP produit et utilisons le plus grand ensemble de transitions locales temporisées exploitables à chaque instant pour produire un chemin jusqu'à ce qu'un attracteur singleton (i.e., point fixe) soit détecté. Cet état global qui est identifié comme un point fixe est discrétisé en inverse (c'est-à-dire le passage du discret au continue) et proposé comme état global prédit. Rappelons que la discrétisation est la transformation des valeurs réelles d'expression des composants biologiques en des valeurs entières. Ainsi, la discrétisation en inverse est la transformation des valeurs d'expression discrètes en des valeurs réelles. En effet, les métriques qu'ils proposent pour évaluer le résultat prennent des expressions des gènes présentées sous la forme des réels.

Dans le challenge, la qualité de la prédiction est évaluée en calculant l'erreur quadratique moyenne (*Mean Square Error* : MSE) entre l'état prédit et l'état attendu. Comme le montre le tableau 6.1 ci-dessus, la précision que nous avons obtenue dans ces expériences est assez bonne compte tenu des résultats des gagnants du challenge DREAM4 (Prill et al., 2011). Leurs résultats se situent entre 0,010 et 0,075 pour les mêmes paramètres d'évaluation, qui sont comparables à de 0,033 à 0,086 dans notre cas (voir tableau 6.1). Ceci nous donne des résultats encourageants. En ce qui concerne le temps d'exécution,

l'apprentissage et la prévision des chemins des modèles des benchmarks de 10 gènes a pris moins de 30 secondes, et les mêmes expériences pour les modèles des benchmarks de 100 gènes ont pris environ 3 heures et 20 minutes avec un processeur Intel Core2 Duo (P8400, 2,26 GHz).

Pour obtenir ce score, nous avons dû effectuer plusieurs tests en faisant varier la précision de discrétisation et la complexité de la dynamique apprise (c'est-à-dire les T-AN appris). La figure 6.4 ci-dessous montre le score obtenu en prédisant les données d'entrainement des modèles des benchmarks du DREAM4 de taille 10 à partir de 2 niveaux de discrétisation jusqu'à 20. Les données d'entrainement sont les données utilisées pour l'apprentissage des modèles. Ici, nous prenons la série de chaque point du benchmark comme entrée, et le premier état global de chaque série est utilisé pour commencer une prédiction. Le MSE est calculé sur les 21 points de la série originale par rapport à ceux qui sont prédits. Nous pouvons voir que l'augmentation de la précision de la discrétisation améliore la précision de la prédiction jusqu'à 5 ou 6 niveaux de discrétisation. Ensuite, la qualité de prédiction du modèle diminue car elle tend à superposer les données. Ces tests nous permettent également d'évaluer le passage à l'échelle de notre approche dans la pratique. La figure 6.5 ci-dessous montre l'impact de la modification de l'indegree (i.e., le nombre des régulateurs par composant dans les transitions locales temporisées apprises) et le choix des niveaux de discrétisation sur le temps d'exécution lors de l'apprentissage des benchmarks de la taille 100.

Dans les résultats obtenus à partir de ces expériences de notre algorithme, `MoT-AN` sur les données de séries temporelles du DREAM4, nous pouvons voir dans la figure 6.5 ci-dessous, que pour les 5 différents modèles appris, l'évolution du temps d'exécution est d'ordre exponentiel avec l'augmentation de l'indegree par transition locale temporisée, ainsi que par niveau de discrétisation choisi. Cependant, cela montre également que, dans la pratique, notre approche peut s'attaquer aux grands réseaux ; ici, de 100 gènes.

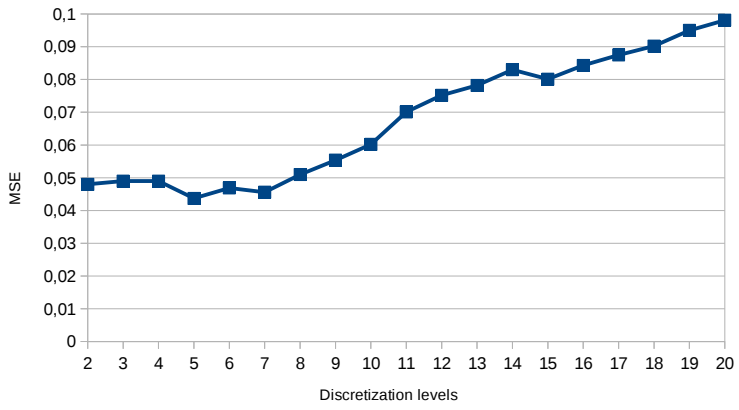


Figure 6.4 : L'impact de l'augmentation des niveaux de la discrétisation sur la précision des modèles appris par la méthode `MoT-AN`. Ces expériences sont faites sur les données d'entraînement des benchmarks du DREAM4 dont la taille est égale à 10 gènes.

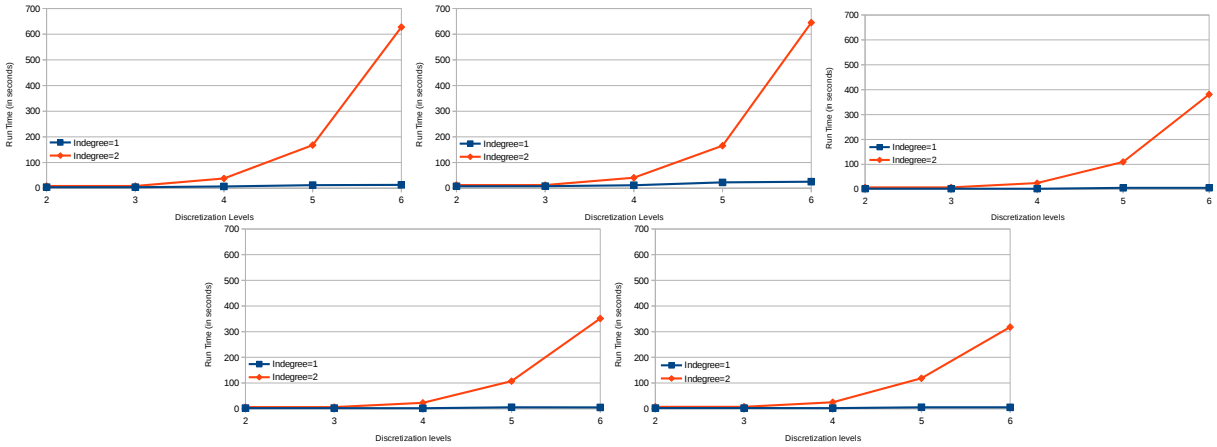


Figure 6.5 : Évolution du temps d'exécution sur le traitement de différents modèles appris à partir des données de séries temporelles de DREAM4 (les benchmarks de taille égale à 100 gènes), en variant l'indegree des transitions locales temporisées et des niveaux de discrétisation. Ces tests sont effectués sur un processeur Intel Core i7 (4700, 3 GHz) avec 16 Go de RAM.

Il est important de noter que, ici, le graphe d'influences est donné en tant que connaissances de base, donc cela réduit considérablement la quantité de transitions locales temporisées apprises pour expliquer chaque changement d'état local d'un composant dans la dynamique. Sans connaissance des influences, l'algorithme 1, `MoT-AN` peut être exécuté en considérant que tous les gènes peuvent être influencés par tous les autres gènes. Cependant, alors, la quantité de combinaisons d'influences est tellement grande que le traitement d'une seule série temporelle discrétisée en 2 niveaux discrets avec un indegree égal à 2 prend plus de 12 heures à traiter. En pratique, sur les grands modèles (plus de 40 gènes), l'accès à l'information sur les influences des gènes (des *knockouts* ou des perturbations) est crucial pour l'efficacité de cette méthode d'apprentissage `MoT-AN`.

6.3.3.2 DREAM8

Nous avons aussi testé notre méthode sur les données de séries temporelles du DREAM8 challenge (Hill et al., 2016). Ce challenge, appelé *Heritage-DREAM*, veut inférer le réseau de régulation lié au cancer du sein[5]. DREAM8 concerne l'inférence des réseaux de signalisation causale et la prédiction de la dynamique de phosphorylation des protéines.

Comme pour le DREAM4, nous nous concentrons sur la partie de la prédiction. L'objectif est de construire des modèles capables de prédire les trajectoires des phosphoprotéines. Un accent important est mis sur la capacité des modèles à généraliser au-delà des données d'entraînement en prédisant des chemins sous des perturbations non vues dans les données d'entrainement. Ce challenge est subdivisé en deux parties indépendantes : (A) des données protéomiques du cancer du sein et (B) des données in-silico. Pour l'apprentissage effectué dans notre cas, nous choisissons de commencer par le premier (A).

Les données d'entraînement (données utilisées pour l'apprentissage)[6] proviennent d'expériences sur 4 lignées cellulaires de cancer du sein stimulées avec différents ligands : MCF7,

[5] L'ensemble des données est disponible en ligne à l'adresse suivante : `https://www.synapse.org/#!Synapse:Syn1720047/wiki/55342`

[6] `Https://www.synapse.org/#!Synapse:syn1720047/wiki/56061`

UACC812, BT20 et BT549 (voir figure 6.6 ci-dessous). Les données d'entrainement sont fournies pour chacun des 32 contextes biologiques définis par la combinaison de la lignée cellulaire et de la condition de croissance (stimulus). Ces données comprennent l'évolution temporelles des expressions pour environ 45 phosphoprotéines et sous différentes perturbations d'inhibiteurs de nœuds de réseau.

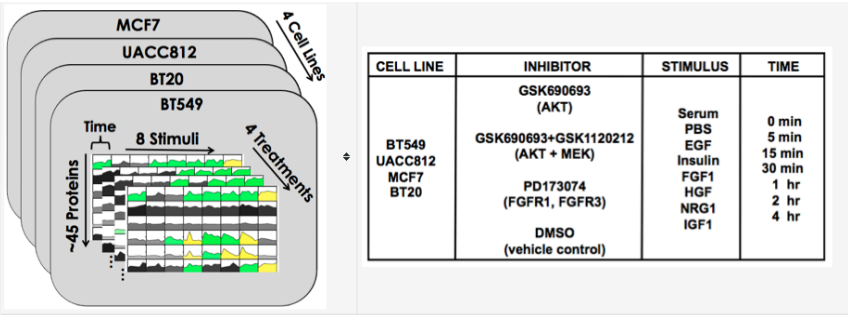


Figure 6.6 : Données de séries temporelles DREAM8 (Hill et al., 2017).

En utilisant ces données d'entraînement pour l'apprentissage des modèles, les participants au DREAM8 challenge sont invités à construire des modèles dynamiques capables de prédire les trajectoires de phosphoprotéines spécifiques telles qu'elles sont sous de nouvelles conditions (des nouvelles perturbations) non fournies dans les données d'entraînement. (elles ne peuvent donc pas être apprises).

Pour évaluer leur précision, les chemins prédits sont comparées avec les données de tests expérimentaux obtenues dans chacun des 32 contextes de lignées cellulaires / stimulus et à la suite de l'inhibition des noyaux de phosphoprotéine par les inhibiteurs de test. Ces expériences sont sur la même période que les données d'entraînement (jusqu'à 4 h). Pour obtenir notre score de précision, nous avons utilisé `dreamtools` (développé par Cokelaer et. al en (2016)), un package Python open-source pour l'évaluation des métriques de notation du DREAM8 challenge. La métrique de notation de ce sous-défi est basée sur plusieurs métriques ; jusqu'à présent, nous nous sommes concentrés sur le carré de la moyenne de l'erreur quadratique (RMSE, Root Mean Squared Error) fournie par `dreamtools`. Nous rappelons que l'erreur est la différence entre ce qui est prédit et ce qui doit être prédit.

Dans ces expériences, nous avons choisi d'abstraire les données en un nombre fixe de niveaux discrets comme pour l'expérience DREAM4 (voir la section 6.3.3.1 précédente). En faisant varier le niveau de discrétisation (voir tableau 6.2 ci-dessous), notre meilleur score est obtenu avec 5 niveaux de discrétisation. Considérer moins de niveaux discrets entraîne une prédiction qui est trop grossière, mais considérer plus de niveaux entraîne une tendance à surajuster le modèle tout en respectant les données prises en entrée.

Le meilleur score que nous avons pu obtenir jusqu'à présent est 0,5528 RMSE qui classerait notre méthode autour des neuvième et onzième[7] qui ont obtenu un score de 0,5139 à 0,5564 avec cette métrique. Dans ce challenge, les méthodes les plus performantes sont basées sur l'analyse statistique[8].

L'intérêt de notre méthode est que le modèle que nous apprenons est lisible par l'être humain, ainsi que la prédiction du chemin. En effet, nous expliquons le comportement

[7]DREAM8 Subchallenge 2A leader board est disponible sur `https://www.synapse.org/#!Synapse:syn1720047/wiki/56831`

[8]voir : `https://www.synapse.org//#!Synapse:syn2343141`

dynamique du réseau en fournissant des interactions détaillées entre les composants pour expliquer chaque changement étape par étape. Ces interactions sont enrichies par des signes (activation / inhibition), le niveau d'expression (seuils) et les délais. Ainsi, nous inférons des transitions locales temporisées dans les T-AN appris. Notre modèle T-AN finalement appris peut être compris statiquement et peut être utilisé pour prédire les comportements dynamiques par rapport à une sémantique connue de la dynamique.

Niveaux de discrétisation	**Temps d'exécution (s)**	**RMSE moyen**
2	9775	0.7054
3	7078	0.6419
4	15,941	0.5901
5	14,102	0.5528
6	19,356	0.5667
7	20,963	0.5563

Table 6.2 : Résultats d'évaluation de notre méthode `MoT-AN` sur les données du DREAM8 Subchallenge 2A selon le RMSE (Root Mean Squared Error).

Comme pour le DREAM4, pour atteindre ce score, nous avons dû effectuer plusieurs tests en faisant varier la précision de la discrétisation. Le tableau 6.3 ci-dessous montre le score du RMSE obtenu en prédisant les données d'entrainement des benchmarks du DREAM8 avec 2 et 5 niveaux de discrétisation. Ici, nous ne montrons que le score détaillé pour deux niveaux, à cause de la contrainte du temps (environ 15 heures par niveau après le niveau 5).

Benchmarks (lignes cellulaires)	**RMSE moyen**	
	2 niveaux de discrétisation	**5 niveaux de discrétisation**
BT20	1.712	0.458
BT549	1.507	0.449
MCF7	0.713	0.310
UACC812	12.6	3.391

Table 6.3 : Résultats d'évaluation de la méthode `MoT-AN` sur les données d'entraînement du DREAM8.

6.4 La vérification de l'atteignabilité

Nous proposons dans cette section, un programme ASP qui vérifie la propriété l'atteignabilité d'un objectif à partir d'un état global initialement donné. Les objectifs peuvent être un état local d'un automate du réseau, un ensemble d'états locaux de différents automates ou encore un état global du réseau.

Nous rappelons que l'étude théorique de cette propriété et l'algorithme de l'approche que nous proposons pour la vérifier sont décrits dans la section 5.3 en page 142 du chapitre 5. Ainsi, le travail présenté dans cette section est principalement fondé sur ce qui a été présenté dans le chapitre 5.

Nous indiquons que nous avons écrit des programmes ASP distincts pour le calcul de la dynamique d'un AN (c'est-à-dire le calcul des chemins) et d'autres pour l'analyse des propriétés dynamiques des AN (c'est-à-dire la vérification de l'atteignabilité et l'identification des attracteurs). Nous montrons dans la section 6.4.1 comment calculer la dynamique d'un AN selon les deux sémantiques de mise à jour : *asynchrone* et *synchrone*. Nous donnons aussi un programme ASP qui calcule la dynamique d'un T-AN selon la sémantique introduite dans le chapitre 3. Il faut noter que le programme qui vérifie la propriété d'atteignabilité, donné dans la section 6.4.2, est commun à toutes les sémantiques de la dynamique.

6.4.1 La dynamique des AN et des T-AN en ASP

Pour analyser un AN en ASP, nous avons d'abord besoin de décrire le réseau en règles logiques d'ASP. Dans ce but, nous avons développé un convertisseur appelé `AN2ASP`[9]. Pour ce processus, nous utilisons le prédicat `automatonLevel` pour définir chaque automate avec ses états locaux (comme c'est le cas pour leur définition dans la méthode de l'apprentissage des T-AN, section précédente 6.3).

L'exemple suivant montre comment un modèle AN est défini en ASP.

Exemple. [Représentation d'un modèle AN en ASP] La définition en ASP des automates du modèle AN de la figure 5.1 en page 134 est la suivante :

```
142 % Les automates et leurs etats locaux
143 automatonLevel("a",0..1). automatonLevel("b",0..2).
144 automatonLevel("c",0..1). automatonLevel("d",0..2).
145 % Les identifiants des automates
146 automaton(A) ← automatonLevel(A,_).
```

Dans les lignes 143–144, nous énumérons tous les automates avec leurs états locaux. Par exemple, l'automate `"a"` possède deux états locaux : a_0 et a_1 introduits par le prédicat `automatonLevel("a", 0..1)`. En effet, ce dernier se développera dans les deux prédicats suivants :

```
automatonLevel("a", 0). automatonLevel("a", 1).
```

À la ligne 146, nous définissons par le prédicat `automaton` les noms des automates existants (qui correspondent aux noms des composants du RRB qu'ils représentent).

Chaque transition locale τ est représentée par deux prédicats : (1) le prédicat `condition` qui définit tous les états locaux de $\mathsf{cond}(\tau)$ ainsi que son origine ($\mathsf{ori}(\tau)$), et (2) le prédicat `target` qui définit la destination de τ ($\mathsf{dest}(\tau)$). Nous notons que chaque transition locale est étiquetée par un identifiant tel que ce dernier est utilisé pour distinguer ses prédicats `condition` et `target`. Par exemple, à la ligne 148, on définit la transition locale $a_0 \xrightarrow{\{c_1\}} a_1$, qui est étiquetée par 1.

Nous déclarons autant de prédicats de `condition` que nécessaire pour définir complètement une transition locale τ qui a potentiellement plusieurs éléments dans sa condition ($\mathsf{cond}(\tau)$). Par exemple, la transition $b_0 \xrightarrow{\{c_1,a_1\}} b_2$ est définie en ligne 152 avec le label 4 et elle nécessite trois instances de ce prédicat : `condition(4,"b",0)` `condition(4,"c",1)` et `condition(4,"a",1)`.

La ligne 164 définit les labels des transitions locales existantes.

[9] Tous les programmes et les benchmarks sont disponibles à l'adresse suivante : `http://www.irccyn.ec-nantes.fr/~benabdal/attractors.zip`

```
147 % Local transitions on a
148 condition(1,"a",0). target(1,"a",1). condition(1,"c",1).
149 condition(2,"a",1). target(2,"a",0). condition(2,"b",2).
150 % Local transitions on b
151 condition(3,"b",0). target(3,"b",1). condition(3,"d",0).
152 condition(4,"b",0). target(4,"b",2). condition(4,"c",1). condition(4,"a",1).
153 condition(5,"b",1). target(5,"b",2). condition(5,"c",1).
154 condition(6,"b",2). target(6,"b",0). condition(6,"c",0).
155 % Local transitions on c
156 condition(7,"c",0). target(7,"c",1). condition(7,"a",1). condition(7,"b",0).
157 condition(8,"c",1). target(8,"c",0). condition(8,"d",2).
158 % Local transitions on d
159 condition(9,"d",0). target(9,"d",1). condition(9,"b",2).
160 condition(10,"d",0). target(10,"d",2). condition(10,"a",0). condition(10,"b",1).
161 condition(11,"d",1). target(11,"d",0). condition(11,"a",1).
162 condition(12,"d",2). target(12,"d",0). condition(12,"c",0).
163 % Les identifiants des transitions locales
164 transition(T) ← target(T,_,_).
```

Nous indiquons qu'en ASP, le symbol '_' dans les paramètres d'un prédicat est un espace réservé pour toute valeur (autrement dit, quelle que soit la valeur qui le remplace). Rappelons aussi qu'un mot commençant par une majuscule est une variable. Puisque la conversion d'un modèle AN en un AN décrit en ASP se fait automatiquement par `AN2ASP` et parfois les noms des composants biologiques commencent par une majuscule, il est préférable d'utiliser des guillemets (`""`) autour des noms d'automates pour s'assurer que les noms des automates sont compris en tant que constantes du programme.

Les réseaux d'automates avec le temps (T-AN) en ASP :

La différence entre un T-AN et un AN est que les transitions locales dans un T-AN sont temporisées. Autrement dit, chaque transition locale temporisée τ dans un T-AN est définie non seulement par $\mathsf{cond}(\tau)$, $\mathsf{ori}(\tau)$ et $\mathsf{dest}(\tau)$ mais aussi par $\mathsf{delai}(\tau)$. Ainsi, nous ajoutons dans la définition de chaque transition locale temporisée en ASP le prédicat `delay(T,D)` qui définit le délai `D` d'une transition étiquetée par `T`.

Par exemple, soit $\tau = a_0 \xrightarrow[3]{\{c_1\}} a_1$ étiquetée par 1 et définie ainsi en ASP :

```
165 condition(1,"a",0). target(1,"a",1). condition(1,"c",1). delay(1,3).
```

L'évolution dynamique des AN :

Nous présentons dans la suite les programmes ASP qui permettent d'énumérer à partir d'un état global initial, tous les chemins de longueur $n \in \mathbb{N}$ dans un AN (définition 5.6 en page 138).

Nous notons que dans un programme ASP, il est possible d'instancier des constantes dont les valeurs sont définies par l'utilisateur à chaque exécution du programme. C'est le rôle de la constante `n` dans le prédicat `step(0..n)` (ligne 168). En effet, `n` représente le nombre maximal des étapes considérées pour calculer l'évolution. Par exemple, si `n=5`, alors `step(0..5)`. Ainsi, sachant l'état global initial, le but du programme ASP est de calculer tous les chemins de longueur 5 (c'est-à-dire après 5 étapes) tels que chaque chemin visite alors jusqu'à 6 états globaux.

Ainsi, afin de spécifier cet état global initial (c'est-à-dire à l'étape 0), nous définissons l'ensemble de tous les états locaux qui y sont actifs. Par exemple, pour le modèle AN de la figure 5.1 en page 134, l'état global initial est $\langle a_1, b_2, c_0, d_1 \rangle$ et est défini en ASP comme c'est indiqué ci-dessous :

```
166 active(level("a",1),0). active(level("b",2),0). active(level("c",0),0). active(level("d",1),0).
```

Ensuite, pour l'identification des successeurs d'un état global (initial ou non initial), il est nécessaire d'identifier tout d'abord l'ensemble des transitions locales qui y sont jouables. En effet, le changement d'un état global vers un autre est réalisé par l'activation des transitions locales qui y sont jouables. Nous rappelons qu'une transition locale est jouable dans un état global quand l'état local de son origine et tous les états locaux des automates dans sa condition sont actifs dans cet état global (voir définition 5.2 en page 135).

Par conséquent, nous définissons par le prédicat `unPlayable(T,S)` à la ligne 170, toute transition locale étiquetée par `T` qui n'est pas jouable à l'étape `S` (c'est-à-dire dans l'état global du réseau à l'étape `S`). Ce qui est le cas lorsqu'au moins l'un des états locaux de sa condition n'est pas actif.

```
167 % Definir toute les etapes d'un chemin de 0 a n
168 step(0..n).
169 % Calculer les transitions locales non jouables a chaque etape
170 unPlayable(T,S) ← active(level(A,I),S), condition(T,A,J), I!=J, step(S).
```

De toute évidence, si une transition locale n'est pas non jouable, alors elle est jouable. Ainsi, on peut trouver les transitions locales qui peuvent être activées parmi celles qui sont jouables. En revanche, le choix d'activer une transition locale jouable ou pas est fait selon la sémantique de mise à jour de la dynamique qui est suivie par le AN. En effet, l'évolution d'un état global vers un autre est faite par les transitions globales. Et ces derniers sont des ensembles de transitions locales et leur calcul dépend principalement de la sémantique de mise à jour de la dynamique. Ainsi, nous présentons dans la suite, les sémantiques de mise à jour qui ont été introduites dans le chapitre 5 précédent : l'asynchrone et le synchrone.

Toutes les évolutions possibles d'un AN (c'est-à-dire tous les chemins trouvés après l'activation successive d'un ensemble de transitions globales) peuvent être énumérées avec une règle de cardinalité. En l'occurrence, celle de la ligne 174 pour la sémantique de mise à jour asynchrone, et celle de la ligne 180 pour la sémantique de mise à jour synchrone. Comme expliqué dans la section 6.2 en page 165, une règle de cardinalité (contient des accolades (`l{...}u`) crée autant d'ensembles de réponses que possibles tels que chaque ensemble respecte la règle, et dont la taille est comprise entre les limites fixées `l` et `u`. Dans notre cas, de telles règles créent autant de réponses qu'il y a de possibles successeurs à partir de chaque état global considéré, reproduisant ainsi tous les chemins possibles dans la dynamique du modèle. Cette énumération englobe le comportement non déterministe (dans les deux sémantiques de mise à jour de la dynamique).

Pour appliquer la dynamique strictement asynchrone qui nécessite qu'un seul automate change entre deux états globaux successeurs ; nous utilisons la contrainte de la ligne 177 pour supprimer tous les chemins où deux ou plus de transitions locales sont activées simultanément et la contrainte de la ligne 176 pour supprimer tous les chemins dans lesquels aucune transition locale n'a été jouée. Ainsi, tous les chemins restants (c'est-à-dire tous les ensembles de réponses restants) suivent strictement la dynamique asynchrone donnée dans la définition 5.3 en page 136. Rappelons que le symbole (`'_'`) dans les

paramètres d'un prédicat est un espace réservé pour n'importe quelle valeur. Ici, il est utilisé à la place de l'étiquette de la transition locale, ce qui signifie que ces règles s'appliquent à toute transition locale.

```
171 % Asynchrone
172 % A chaque etape, calculer et generer toutes les transitions locales activees
173 % dans les differents ensembles de reponses
174 { played(T,S) } ← not unPlayable(T,S), transition(T), step(S).
175 % Exactement une transition locale est activee
176 ← 2 { played(_,S) }, step(S).
177 ← 0 { played(_,S) } 0, step(S).
```

La deuxième sémantique de mise à jour de la dynamique d'un AN étudiée est la sémantique synchrone. Dans cette sémantique, toutes les transitions locales qui sont jouables (et qui ne sont pas en conflit) doivent être activées simultanément (voir définition 5.4 en page 136).

```
178 % Synchrone
179 % A chaque etape, calculer toutes les transitions locales activees
180 1 { played(T,S) } ← not unPlayable(T,S), transition(T), step(S).
181 % A chaque etape, s'il existe une transition locale jouable alors elle doit etre activee
182 ← 0 { played(_,S) } 0, step(S).
```

En bref, il faut choisir l'un des deux programmes ASP présentés ci-dessus, soit les lignes 174–177 pour la sémantique asynchrone, soit les lignes 180–182 pour le synchrone. Le résultat de ces deux programmes est une collection d'ensembles de réponses, avec une réponse pour chaque chemin possible de longueur `n` (c'est-à-dire calculée en `n` étapes) et à partir d'un état initial donné (correspondant à l'étape `0`). Nous notons que plusieurs autres sémantiques de mise à jour de la dynamique sont possibles ; il suffit de modifier les définitions des contraintes pour définir d'autre sémantiques.

Entre deux étapes successives `S` et `S+1`, si une transition locale est activée dans l'état global de `S`, alors un changement doit être observé dans l'état global de `S+1`. Autrement dit, l'automate concerné doit changer son état local actif vers l'état local de la destination de celui de la transition locale activée (ligne 188) sinon il garde le même état local (ligne 189).

Ainsi, nous trouvons tous les automates qui subissent le changement d'un état local actif vers un autre avec le prédicat `change` de la ligne 184.

Rappelons que les transitions locales qui sont en concurrence (voir définition 5.9 en page 141) ne peuvent pas être activées en parallèle. En effet, elles font évoluer le même automate vers des états locaux différents, ce qui n'est pas possible car chaque automate ne peut avoir au plus qu'un état local actif dans chaque état global. Nous ajoutons ainsi la contrainte de la ligne 186, qui indique qu'au maximum un changement peut se produire (i.e., une seule transition locale peut être activée) dans le même automate.

```
183 % Etat local de A change de I vers J
184 change(A,I,J,S) ← played(T,S), target(T,A,J), condition(T,A,I).
185 % Exactement un changement par automate (transition en concurrence)
186 ← X={change(_,A,I,_,S)}, step(S), automaton(A), X>1.
187 % Calculer le nouvel etat global dans S+1 (successeur de S)
188 active(level(B,K),S+1) ← not change(_,B,_,_,S), active(level(B,K),S), step(S), S<n.
189 active(level(B,K),S+1) ← change(_,B,_, K, S), S<n.
```

L'évolution de la dynamique des T-AN :

Nous introduisons dans la suite, l'implémentation en ASP de la méthode qui calcule l'évolution de la dynamique des T-AN selon la sémantique de la dynamique qui est introduite dans la section 3.3 en page 64 du chapitre 3.

Nous rappelons brièvement que pour les T-AN, le changement dans automate d'un état local vers un autre est effectué par une transition locale temporisée qui n'est pas instantanée. Autrement dit, si une transition locale temporisée τ est activée à une étape S, alors le changement de l'automate auquel elle appartient n'est lisible qu'à l'étape $S + D$ avec $D = \mathsf{delai}(\tau)$. Et pendant la période de temps entre S et $S + D$, τ est considérée en cours d'activation (en ASP, on la définit par le prédicat `processingTrans` à la ligne 207).

Ainsi, toute transition locale temporisée qui modifie l'une des ressources de τ (c'est-à-dire de $\mathsf{cond}(\tau)$) est bloquée pendant ce temps (entre S et $S + D$). C'est-à-dire elle est considérée comme non jouable et en ASP, on la note par le prédicat `unPlayable` (lignes 198–199).

En plus, toute transition locale temporisée qui nécessite que l'automate, qui est en train d'être changé par τ reste au même état local actif, jusqu'à une étape supérieure à $S + D$ (i.e., l'étape à laquelle τ se termine), cette transition est bloquée. Ainsi, elle est aussi identifiée en ASP par le prédicat `unPlayable` (lignes 201–202).

Les autres cas où une transition locale temporisée est considérée comme non jouable, c'est quand au moins l'une de ces conditions n'est pas validée (lignes 192–193). Ou encore si cette transition a été activée à des étapes précédentes et qu'elle est en cours d'activation (ligne 195).

```
190 % Calculer les transitions locales temporisees non jouables a chaque etape :
191 % si l'une des conditions de la transition T n'est pas verifiee
192 unPlayable(T,S) ← active(level(A,I),S), condition(T,A,J), I!=J, change(S),
193         not processingTrans(T,S).
194 % ou si la transition T est en cours d'activation
195 unPlayable(T,S) ← processingTrans(T,S), change(S).
196 % ou si la transition T est bloquee par T1 a cause du partage de la meme ressource A
197 % A ne peut pas etre modifiee car c'est une ressource pour la transition en cours T1
198 unPlayable(T,S) ← condition(T,A,I), target(T,A,_), protected(A,I,T1,S), clock(T1,Dc,S),

199         D<D1-Dc, T!=T1, delay(T,D), delay(T1,D1).
200 % L'automate A n'est pas valable pendant toute la periode d'activation de T (change par T1)
201 unPlayable(T,S) ← condition(T,A,I), target(T1,A,_), clock(T1,Dc,S), D>D1-Dc,
202         T!=T1, delay(T,D), delay(T1,D1),.
```

Et donc, une transition locale temporisée qui est jouable et qui est activées à une étape `S` est identifiée par le prédicat `playable` (ligne 204). Ainsi, pendant la période de temps entre l'instant où une transition est activée et l'instant auquel elle se termine, cette transition est considérée en cours d'activation : dans le programme ASP, elle est identifiée par le prédicat `processingTrans`. Ainsi, chaque automate appartenant à sa condition (donc nécessaire pour son activité) ne peut pas être modifié pendant sa période d'activation. Nous identifions ces automates par le prédicat `protected` (ligne 210).

```
203 % Calculer les transitions locales temporisees jouables a chaque etape S
204 1 { playable(T,S) }  ← not unPlayable(T,S), transition(T), change(S), S>1.
205
206 % Transition en cours d'activation
```

```
207 processingTrans(T,S) ← playable(T,S1), delay(T,D), S<S1+D, S>S1, step(S).
208
209 % Les automates qui ne peuvent pas etre changes a une etape S
210 protected(A,I,T,S) ← processingTrans(T,S), condition(T,A,I).
```

Nous identifions les instants auxquels il y a eu des changements des états locaux des automates par le prédicat `change(S)` (lignes 229–232). C'est seulement à ces étapes que nous identifions les transitions locales temporisées qui sont jouables, ou nouvellement jouables car il y a eu un changement d'état. En effet, ceci réduit la complexité de la recherche puisqu'on n'est pas obligé de les identifier à chaque étape mais seulement s'il y a un changement dans la dynamique (car il y a un automate qui change d'état local actif). C'est le prédicat `change(S)` (lignes 229–232) qui indique chaque étape `S` à laquelle un changement s'est produit.

Nous considérons que l'état global du réseau est initialement donné par le prédicat `active(level(A,I),0)` qui est défini pour chaque automate `A` du réseau (avec `I` son état local actif à l'étape 0). À chaque étape `S`, et pour chaque transition locale temporisée `T`, nous définissons par le prédicat `clock(T,Dc,S)` l'*horloge* qui sauvegarde la durée du temps écoulée, `Dc`, pendant laquelle la transition `T` est en cours d'activation. Ainsi, si à une étape `S`, `Dc` atteint la valeur de `D`, qui est le délai de `T` (avec `delay(T,D)`), alors l'activation de la transition locale temporisée `T` se termine à cette étape `S`. Par conséquent, cette fin d'activation implique le fait que l'horloge de `T`, dans `clock(T,Dc,S)`, est remise à 0 (ligne 221) et que l'automate ciblé par la transition `T` change l'état local actif ; de `I` vers `J` selon `condition(T,A,I)` et `target(T,A,J)` (voir ligne 229). Finalement, le nouvel état global du réseau est calculé à chaque étape `S` par les lignes 239–241.

```
211 % n est une constante du programme et elle est egale a la taille maximale du chemin
212 step(0..n).
213 % A l'etape 0, nous initialisons a 0 la periode du temps ecoulee de l'activation de chaque
214 % transition locale temporisee
215 clock(T,0,0) ← transition(T).
216 % Si la transition est activee a l'etape S, alors l'horloge est egale a 1 a S+1
217 clock(T,1,S) ← playable(T,S), step(S).
218 % L'horloge s'incremente tant que la transition est en cours et qu'elle n'atteint pas le delai
219 clock(T,Dc+1,S+1) ← clock(T,Dc,S), processingTrans(T,S), delay(T,D), Dc<D.
220 % L'horloge est remise a 0 si le delai est atteint
221 clock(T,0,S+1) ← clock(T,D,S), delay(T,D).
222 % L'horloge garde la valeur 0 si la transition n'est pas activee
223 clock(T,0,S+1)  ← not processingTrans(T,S), not change(T,S), not playable(T,S),
224         step(S), transition(T).
225
226 % Identifier les changements des etats locaux des automates
227 % L'automate A change du niveau I vers J a l'etape S car l'horloge atteint D
228 % qui est le delai de la transition T
229 change(T,A,I,J,S) ← clock(T,D,S), delay(T,D), condition(T,A,I), target(T,A,J).
230 change(A,I,J,S) ← change(_,A,I,J,S).
231 change(T,S) ← change(T,_,_,_,S).
232 change(S) ← change(_,_,_,_,S).
233
234 % Tout ensemble de reponses illustre au moins un changement dans la dynamique
235 ← X={change(_)}, X=0.
236
237 % Calculer l'etat global du T-AN a chaque etape S : S est le successeur de S-1
238 % L'automate A garde son etat local I s'il ne change pas
239 active(level(A,I),S) ← active(level(A,I),S-1), not change(_,A,_,_,S), step(S), S>0.
240 % L'automate A change vers un nouvel etat local J
241 active(level(A,J),S) ← change(_,A,_,J, S), S>0.
```

Finalement, on dit que quel que soit le formalisme utilisé (AN ou T-AN) et quelle que soit la sémantique de mise à jour de la dynamique, les programmes développés ci-dessus retournent l'évolution de la dynamique du réseau étudié. En effet, ces programmes logiques retournent l'évolution des états locaux actifs de chaque automate au cours du temps. Ceci est récupéré par le prédicat `active(level(A,I),S)`. À chaque étape `S` (i.e., `step(S)`), on a l'état local `I` de chaque automate `A` du réseau. Par conséquent, nous ajoutons une dernière règle à la ligne 242 ci-dessous pour tous les programmes ASP qui calculent l'évolution de la dynamique des AN et des T-AN. Cette règle à la ligne 242 permet de n'afficher dans chaque ensemble de réponse que le prédicat `active` qui indique l'état local actif de chaque automate du réseau à chaque étape.

```
242 #show active/2.
```

Pour vérifier alors si un état local est atteignable à partir d'un état global initial donné, nous prenons les ensembles de réponses retournés par les programmes ASP décrits ci-dessus qui calculent l'évolution de la dynamique du réseau. Ensuite, nous appelons un autre programme ASP (présenté dans la section suivante) qui permet de vérifier dans tous ces chemins –qui sont énumérés de façon exhaustive– s'il en existe un ou plusieurs qui vérifient la propriété d'atteignabilité des objectifs choisis.

6.4.2 L'atteignabilité en ASP

Dans cette section, nous développons un programme ASP pour la vérification de l'atteignabilité d'un ensemble d'états locaux des automates à partir d'un état global initial. Nous rappelons que cette propriété d'atteignabilité est étudiée sur le plan théorique dans la section 5.3 en page 142 du chapitre 5. Brièvement, l'étude de la propriété d'atteignabilité se résume par la réponse à la question suivante : "est-il possible, à partir d'un état global initial donné, d'activer successivement un certain nombre de transitions afin qu'un ensemble d'états locaux donnés soient atteints dans l'état global résultant ? " Nous utilisons la mise en œuvre du calcul de la dynamique donné dans la sous section 6.4.1 précédente, afin de résoudre ce problème d'atteignabilité.

Alors, nous définissons premièrement, un prédicat `goal` pour énumérer les états locaux des automates que nous voulons atteindre par la dynamique du réseau. Par exemple, dans la règle en ligne 243, nous définissons l'état local z_1 comme objectif. Et il est possible d'ajouter autant de règles de ce genre qu'il y a d'états locaux comme objectif.

```
243 goal(level("z",1)).
```

Le prédicat `unReached(Lv,S)` (ligne 245) identifie les objectifs qui ne sont pas encore atteints à une étape `S`. Et donc pour vérifier, si à une étape `S`, tous les objectifs sont atteints nous utilisons la règle à la ligne 247 telle que `reached(S)` est vrai seulement si tous les objectifs définis dans `goal(Lv)` sont atteints. Finalement, si, dans un chemin illustrant l'évolution de la dynamique du réseau, il n'existe aucune étape à laquelle tous les objectifs sont atteints, alors ce chemin est à éliminer (ligne 250).

```
244 % les objectifs non atteints
245 unReached(Lv,S) ← goal(Lv), not active(Lv, S), step(S).
246 % L'etape a laquelle tous les objectifs sont atteints
247 reached(S) ← step(S), not unReached(Lv, S), goal(Lv)
248 reached ← reached(S).
249 % les objectifs ne sont atteints a aucune etape
250 ← not reached.
```

Ainsi, quand nous appelons ce programme qui vérifie l'atteignabilité avec le programme qui calcule les chemins de l'évolution du réseau, seulement les chemins qui permettent d'atteindre nos objectifs fixés sont retournés dans les ensembles de réponses.

Optimisation

La limitation de la méthode ci-dessus est que l'utilisateur doit décider préalablement du nombre d'étapes à traiter (c'est-à-dire le choix de la constante n des programmes ASP qui calculent les chemins de longueur n). Cette constante devrait être alors suffisamment grande pour que nous soyons sûrs d'atteindre tous les objectifs. Autrement dit, la vérification est bornée par n. C'est un inconvénient principal qui est partagé par exemple par la méthode proposée dans (Rocca, Mobilia, Fanchon, Ribeiro, Trilling & Inoue, 2014) qui vérifie les propriétés CTL dans les modèles de Thomas.

Une solution consiste alors à utiliser un mode de calcul incrémentiel, qui est particulièrement abordé par le solveur incrémental de Clingo (Gebser, Sabuncu & Schaub, 2010). La syntaxe correspondante subdivise le programme en 3 parties. La partie `#program base` et qui ne contient que des éléments non incrémentaux et est donc utilisée pour déclarer des règles générales qui ne dépendent pas des étapes des instants temporels. Le programme ASP qui doit être fait pour chaque itération est placé dans les parties `#program step(s)`

et `#program check(s)`, qui sont calculés à chaque étape incrémentielle `s`. Notez que le numéro d'étape `s` n'est pas une variable mais une constante pour chaque itération. La partie `#program step(s)` comprend des règles dépendantes des étapes dans l'évolution de la dynamique (similaire au prédicat `step` que nous avons déclaré précédemment à la ligne 168 et ligne 212), et la partie de `#program check(t)` contient les contraintes qui arrêtent l'itération si nécessaire (en occurrence dans notre cas, quand l'objectif est atteint).

Lors de l'utilisation de cette nouvelle syntaxe, le programme ASP obtenu est presque identique à ce qui a été présenté avant, sauf que les numéros d'étapes notées par `S` sont remplacés par la constante `s`. Ainsi, une légère modification sera faite dans les programmes ASP qui calculent l'évolution de la dynamique d'un AN et d'un T-AN ; à chaque occurrence, remplacer la variable `S` par la constante `s`.

Finalement, le solveur compare alors ses ensembles de réponses avec la contrainte qui dépend de `s` et qui est donnée dans la partie de `check(s)`. En ce qui concerne notre implémentation, cette contrainte, donnée dans les lignes 252–253, affirme simplement que tous les objectifs doivent être atteints. Si cette contrainte invalide tous les ensembles de réponses courants, le calcul continue pour la prochaine itération (i.e., `s+1`) afin d'atteindre un ensemble de réponses valides. Dès qu'un ensemble de réponses n'est pas filtré par la contrainte (c'est-à-dire l'objectif est atteint), il est retourné et le calcul s'arrête. Par conséquent, cette méthode permet de retourner le plus court chemin qui permet d'atteindre tous les objectifs.

```
251 #program check(s).
252 unReached(s) ← goal(Lv), not active(Lv,s).
253 ← unReached(s), query(s).
```

Élimination des boucles

La version itérative de notre programme ASP qui résout la problématique de l'atteignabilité, itérerait indéfiniment si la dynamique d'un modèle contient une boucle. Pour pallier cela, nous ajoutons dans la partie de `#program step(s)` des règles logiques qui éliminent toute réponse (i.e., tout chemin) dans lequel le réseau cycle sur un même état global sans arrêt. Ainsi, nous assurons qu'au cours de toute l'évolution de la dynamique, tous les états globaux visitées sont tous différents.

```
254 time(0..s-1).
255 % les etats globaux aux etapes S et s sont differents s'il y a au moins un automate
256 % ayant un etat local actif different dans S et s
257 different(s,S,A) ← active(level(A,I),s), active(level(A,J),S), I!=J, time(S), automaton(A).
258 % Il y a un cycle entre S et s quand il n'y a aucune difference entre elles
259 loop(s,S) ← not different(s,S,_).
260 % Eliminer cette reponse
261 ← loop(s,_).
```

6.4.3 Applications

Dans cette section, nous montrons l'efficacité de notre approche pour la vérification de l'atteignabilité par son application sur quelques exemples des AN issus de réseaux biologiques réels. Dans la suite, nous appelons notre méthode dont le programme ASP est introduit dans la section 6.4.2 précédente par ASP-AN. Nous notons que les expériences sont effectuées

avec l'approche itérative présentée dans la section précédente. Tous les calculs détaillés dans cette section sont effectués sur un Pentium V, 3.2 GHz with 4 GB RAM.

Pour évaluer l'efficacité de notre nouvelle approche, nous nous positionnons par rapport aux méthodes existantes qui traitent différents formalismes de modèles biologiques. Nous avons comparé nos résultats à ceux obtenus avec les outils suivants : GINsim[10] : *Gene Interaction Network Simulation* (Gonzalez, Naldi, Sanchez, Thieffry & Chaouiya, 2006) ; libDDD[11] : (Colange, Baarir, Kordon & Thierry-Mieg, 2013) ; Pint[12] (Paulevé, 2016*b*) ; et finalement la méthode proposée par (Rocca et al., 2014) que nous notons par ASP-Thomas qui a également été développée en ASP et qui vérifie les propriétés dynamiques des modèles CTL (*CTL model-cheking*).

Chacune de ces méthodes utilise un formalisme spécifique pour la représentation des RRB[13] ; le modèle de Thomas (un formalisme particulier des réseaux de régulation logiques) est utilisé par GINsim et la méthode de Rocca *et al.*, des systèmes de transitions pour LibDDD, et le formalisme des AN pour Pint et notre méthode ASP-AN.

Les spécifications des modèles utilisés pour les tests sont résumées dans le tableau 6.4 ci-dessous. Les résultats concernant la propriété d'atteignabilité des méthodes (Pint, LibDDD, GINsim et de notre méthode ASP-AN) sont récapitulés dans le tableau 6.5 ci-dessous. Nous discutons dans la suite de ces résultats obtenus par rapport aux autres méthodes. Par contre, globalement, les résultats montrent que notre méthode est efficace pour la vérification de l'atteignabilité.

Modèles	Description des AN		
Nom	Automates	États locaux	États globaux
TTR	12	42	2^{19}
ERBB	42	152	2^{70}
TCR	54	156	2^{73}

Table 6.4 : Modèles utilisés dans nos tests d'atteignabilité. Chaque modèle est désigné par son nom abrégé, où TTR représente le modèle de résorption de queue de têtard (Khalis et al., 2009), ERBB pour la transition G1/S régulée par le récepteur du même nom (Samaga et al., 2009) et TCR pour le réseau de signalisation du récepteur de cellules T (Klamt et al., 2006). Pour chacun d'entre eux, ce tableau donne le nombre d'automates, le nombre d'états locaux et le nombre d'états globaux dans le modèle AN correspondant.

[10] GINsim version 2.4 alpha : `http://ginsim.org/`

[11] LibDDD version 1.8 : `http://move.lip6.fr/software/DDD/`

[12] Pint version 2015-11-14 : `http://loicpauleve.name/pint/`

[13] Quand c'est disponible, nous avons utilisé les convertisseurs inclus dans l'outil Pint pour la traduction entre les différents formalismes.

Expériences			Résultats			
	Modèle	Objectif ω	Pint	LibDDD	GINsim	ASP-AN
#1	TTR	$\in \mathcal{S}$	0.97s	1.15s	2.05s	1.90s
#2	ERBB	$\in \mathcal{S}$	out	1mn55.38s	2mn31.64s	11.84s
#3	ERBB	$\in \wp(\mathbf{LS})$	0.03s	1mn4.96s	–	5.02s
#4	TCR	$\in \mathcal{S}$	Inconc	out	out	6mn27.93s
#5	TCR	$\in \wp(\mathbf{LS})$	0.02s	out	–	1mn35.08s

Table 6.5 : Performances comparées de plusieurs méthodes qui vérifient la propriété de l'atteignabilité : La méthode de Pint, LibDDD, GINsim et notre nouvelle méthode appelée ASP-AN. Pour chaque test, ce tableau donne le nom abrégé du modèle considéré, comme indiqué dans le tableau 6.4 ci-dessus. Le type d'objectif ω est soit un état global (i.e., $\omega \in \mathcal{S}$) soit un ensemble d'états locaux de différents automates (i.e., $\omega \in \wp(\mathbf{LS})$). La colonne des résultats détaille le temps de calcul mis par des différentes méthodes. "out " marque une exécution prenant trop de temps ou de mémoire, "–" indique qu'il n'est pas possible de faire le test, et "Inconc" indique que la méthode se termine sans réponse.

Nous présentons dans la suite une comparaison détaillée des résultats du tableau 6.5 ci-dessus.

- **GINsim** (Gonzalez et al., 2006) :
est un logiciel pour l'édition, la simulation et l'analyse des réseaux d'interactions génétiques. Il permet de vérifier le problème d'atteignabilité par une approche qui consiste à calculer le graphe (ou une partie du graphe) d'états de transitions, puis elle vérifie l'existence d'un chemin entre les deux états globaux donnés.

Par conséquent, il n'est pas possible d'effectuer la vérification d'atteignabilité sur un ensemble d'états locaux (i.e., $\omega \in \wp(\mathbf{LS})$) mais seulement si l'objectiof est un état global (i.e., $\omega \in \mathcal{S}$) ; expériences #3 & #5. Nous notons que cet outil permet d'afficher de petits graphes d'états de transitions.

- **LibDDD** (Colange et al., 2013) :
est une bibliothèque pour la vérification des propriétés CTL & LTL des modèles. Elle peut donc notamment être utilisée pour vérifier la propriété d'atteignabilité ; il s'agit de vérifier la propriété **EF(P)** de la logique temporelle qui exprime qu'il existe un chemin menant fatalement à un état où la propriété **P** est vérifiée.

Cependant, contrairement à notre méthode, elle ne retourne pas un chemin d'activation pour résoudre une atteignabilité. En outre, elle repose sur la construction du graphe d'états de transitions qui est alors stocké sous la forme d'un diagramme de décision binaire pour que l'analyse du graphe soit plus efficace. Ce calcul explique pourquoi LibDDD prend plus de temps pour répondre, et consomme toute la mémoire en environ 12 minutes pour le plus grand exemple qui contient 2^{73} états globaux (expériences #4 & #5).

- **Pint** (Paulevé, 2016*b*) :
est une bibliothèque rassemblant des outils et des convertisseurs liés au formalisme des AN. La vérification de l'atteignabilité réalisée par Pint qui procède par une approximation pour éviter de calculer tout le graphe d'états de transitions. Pint n'est donc pas assuré d'être toujours efficient mais il est très rapide en temps de calcul ; ce qui explique les résultats les

plus rapides (dans le tableau 6.5 ci-dessus), mais au prix d'un éventuel arrêt de calcul sans être concluant.

Cependant, il n'est pas conçu pour des objectifs consistants en de nombreux états locaux, qui sont plus susceptibles de déclencher une réponse non concluante (comme pour l'expérience #4), ou une recherche exponentielle dans les sous-solutions de la vérification et ce qui exige un temps énorme (comme pour l'expérience #2).

- **ASP-Thomas**[14] (Rocca et al., 2014) :
est un programme ASP qui offre la possibilité de modéliser les propriétés CTL sur les réseaux de Thomas. Actuellement, à notre connaissance, la traduction des réseaux de Thomas en ASP, y compris les paramètres discrets, doit être faite à la main. Ensuite, pour une formule CTL donnée également décrite en ASP, ASP-Thomas retourne exhaustivement l'ensemble de chemins satisfaisant la formule. Dans notre comparaison, nous nous concentrons seulement sur l'atteignabilité, c'est-à-dire les formules logiques de la forme : $s_0 \Rightarrow \mathsf{EF}(g)$ où s_0 est l'état initial et g une propriété d'objectif.

Il faut noter que cette méthode nécessite de fournir un nombre n qui est égal au nombre d'étapes pour lequel la dynamique sera calculée. En effet, toutes les dynamiques du système sont calculées jusqu'à n étapes afin d'effectuer une première recherche en *profondeur* (dans le graphe d'états de transitions). Par conséquent, cette valeur n doit être plus grande que la taille du chemin minimal et inférieure à la taille du chemin maximal, et elle peut donc être difficile à prédire. Ceci est aussi le cas pour la version non itérative de notre méthode (avant l'optimisation dans la section 6.4.2 précédente). Par contre, ce n'est pas le cas pour la version itérative de notre méthode qui peut s'arrêter dès que l'objectif est atteint, et donc potentiellement avant la $n^{\text{ème}}$ étape.

La principale conséquence de la méthode ASP-Thomas est que si on peut a priori connaître la longueur du chemin pour atteindre l'objectif spécifié, cette approche peut fournir de très bonnes performances. Cela montre qu'ASP peut être un bon choix pour calculer la dynamique d'un modèle et vérifier les propriétés dynamiques. Toutefois, si la limite est totalement inconnue, le choix d'une longueur de chemin trop petite peut naturellement conduire à manquer l'objectif, et une valeur trop grande aura un grand impact sur la performance ou même conduire à manquer le but (si aucun chemin de longueur n n'existe). Par exemple, dans l'expérience #3, l'objectif est atteint en 18 étapes : en utilisant un n qui est égal à 21, ASP-Thomas finit en 2.61s. Cependant, si n= 30 étapes, il faut plusieurs minutes pour terminer le calcul des chemins de longueur n=30.

Pour conclure, on peut dire qu'avec notre approche itérative, si l'objectif est atteignable, le temps d'exécution du programme dépend uniquement de la longueur du chemin qui permet d'atteindre cet objectif. Alors qu'avec ASP-Thomas, le temps d'exécution dépend principalement de la valeur de n choisie (i.e., la longueur des chemins à calculer) ; car la méthode génère tous les chemins de la longueur choisie avant d'être vérifiés. Un résumé des différences entre notre approche et ASP-Thomas est détaillé dans le tableau 6.6 ci-dessous.

[14]Nous souhaitons remercier Laurent Trilling pour son aide à effectuer ces tests.

	ASP-Thomas	ASP-AN
Modèle	Réseau de Thomas	Réseau d'Automates
Propriété	formule CTL	que EF
Résultat	tous les chemins	chemin minimal
Atteignabilité	borné	**non borné**
Non-atteignabilité	borné	borné
Recherche	en profondeur	en largeur
Temps d'exécution (petit n)	**secondes**	secondes/minutes
Temps d'exécution (grand n)	Out/UNSAT	**secondes/minutes**

Table 6.6 : Comparaison qualitative entre la version optimale de notre approche ASP-AN et la méthode ASP-Thomas.

6.4.4 Discussion

Dans les sous sections précédentes, nous avons développé une nouvelle méthode implémentée en ASP qui vérifie la propriété d'atteignabilité. En effet, elle permet de trouver le plus court chemin pour atteindre un objectif donné à partir d'un état global initial donné.

Nous avons pu conclure à partir des résultats des expériences effectuées et discutées, que par rapport à d'autres méthodes décrites ci-dessus, GINsim et LibDDD, notre méthode est relativement plus rapide et permet également d'étudier des réseaux assez grands (jusqu'à 2^{73} états globaux).

Dans notre étude, nous avons étudié les RRB modélisés avec le formalisme des AN (et dans sa version étendue T-AN). Ce formalisme est une restriction des automates synchrones et permet ainsi de représenter n'importe quel type de système dynamique. De plus, la dynamique des AN est facile à implémenter en ASP (section 6.4.1 en page 187).

En pratique, nous pouvons détecter les boucles dans la dynamique du modèle et éviter de vérifier à nouveau le même chemin. Cependant, la version itérative de Clingo peut itérer à jamais même si aucune réponse n'est satisfaite et il n'y a plus de chemin à explorer. Ainsi, bien qu'il soit prévu de retourner aucune réponse (i.e., `insatisfiable`), le solveur reste coincé à l'infini dans une boucle. À notre connaissance, il n'est pas encore possible de forcer l'incrémentation à s'arrêter dans la version itérative du solveur sans atteindre les objectifs (c'est-à-dire les conditions d'arrêt). En revanche, il est encore possible de limiter le nombre d'itérations à un nombre arbitraire maximal qui sera finalement atteint dans ce cas spécifique (i.e., l'objectif est non atteignable et il y a une boucle dans le réseau). Ceci est possible avec l'utilisation d'une option lors de l'appel du programme ASP qui limite le nombre maximal d'étapes : `"#const imax = k"`, où `k` est le nombre maximum d'étapes (qui n'est pas forcément la longueur du chemin).

Plusieurs valeurs peuvent être données à ce paramètre `k`. Par exemple, le nombre total d'états globaux est un maximum évident, car il ne sera jamais dépassé par un chemin minimum, mais il est trop élevé pour être très intéressant. Le nombre total des automates est une valeur plus intéressante, sous l'hypothèse que chacun changera son état local actif au plus une fois, ce qui est souvent le cas pour les réseaux booléens. Ou encore, avec un raisonnement analogue, `n` peut être égal au nombre total des états locaux dans le réseau.

Compte tenu de notre implémentation, si l'étape `k` est atteinte (ce qui signifie qu'aucun chemin valide n'a été trouvé), le calcul s'arrête avec une réponse `unsatisfiable` pour l'atteignabilité. En revanche, si l'objectif est atteint à une étape intermédiaire (c'est-à-dire inférieure à `k`), il n'est pas nécessaire de continuer l'itération jusqu'à `"k"`. Le calcul est arrêté et le chemin trouvé vérifiant l'atteignabilité est renvoyé.

Dans la section suivante, nous étudions une autre propriété dynamique dans les AN et qui consiste à identifier leur ensemble d'attracteurs.

6.5 La recherche des attracteurs

Nous présentons dans cette section les programmes ASP que nous avons développés en nous fondant sur l'étude théorique faite sur la recherche des attracteurs introduite dans la section 5.4 en page 146 du chapitre 5. Ainsi, dans plusieurs endroits de cette section nous nous référons à des définitions et des propriétés du chapitre 5.

6.5.1 Les attracteurs cycliques en ASP

Rappelons qu'un attracteur est un ensemble d'états globaux tels qu'une fois atteints, il n'est plus possible de s'en échapper. En effet, comme c'est indiqué dans la définition 5.12 en page 148, il s'agit d'un domaine piège minimal.

Dans cette section, nous nous concentrons sur l'identification des attracteurs cycliques (i.e., non singletons). Nous introduisons alors les programmes ASP qui étudient la dynamique des AN et qui permet d'identifier certains ou tous ses attracteurs d'une certaine taille. Nous notons que l'identification de tous les attracteurs de toutes les tailles peut être théoriquement abordé en augmentant progressivement la taille considérée.

Rappelons ci-dessous les 4 étapes de la méthode qui identifie les attracteurs cycliques dans un AN et qui a été présentée dans la section 5.4.4 en page 156 :

1. énumérer tous les chemins de longueur $n \in \mathbb{N}^*$ (n est une constante choisie arbitrairement) ;
2. parmi les chemins énumérés, éliminer tous ceux qui ne sont pas des cycles ;
3. pour chaque trace d'un cycle restant, vérifier qu'il est aussi un domaine piège et donc c'est un attracteur ;
4. vérifier que n est la longueur minimale des chemins qui parcourent tous les états globaux de l'attracteur.

Ainsi, pour identifier les attracteurs, nous suivons dans la suite le même raisonnement (i.e., ces 4 étapes) mais avec des programmes ASP dédiés.

Énumérer des chemins de longueur n en ASP :

Dans la section 6.4.1 en page 187, nous avons défini un programme ASP qui calcule la dynamique des AN (i.e., des chemins) à partir d'un état global initial donné et après un certain nombre d'étapes. Ainsi, nous utilisons ce même programme (lignes 168–189 en pages 189-190) dans cette première étape de la méthode d'identification des attracteurs

dans les AN. Une seule différence qu'on doit noter c'est que pour l'identification des attracteurs, les chemins calculés n'ont pas un état global initialement donné comme c'est le cas des chemins calculés pour la vérification de la propriété d'atteignabilité. En effet, pour être exhaustifs, nous devons considérer tous les états globaux que peut avoir le réseau. Ainsi, nous ajoutons au début du programme ASP une règle (ligne 263) avec une cardinalité dans sa tête qui génère tous les états globaux du AN étudié. Le but est alors de vérifier si à partir de chacun de ces états globaux un attracteur est atteignable.

```
262 % Enumerer tous les etats globaux initiaux a l'etape 0
263 1 { active(level(A,I),0) : automatonLevel(A,I) } 1 ← automaton(A).
```

Par conséquent, ce programme retourne plusieurs ensembles de réponses tels que chaque ensemble présente un chemin de longueur `n` et qui sont calculés à partir de différents états globaux initiaux. L'étape suivante consiste alors à n'en garder que les cycles.

Éliminer tous les chemins qui ne sont pas des cycles :

Après la construction d'un chemin de longueur `n`, il faut vérifier s'il s'agit d'un cycle (et par conséquent d'une composante fortement connexe, voir lemme 5.2 en page 149) ou non. Pour cela, nous proposons un programme ASP qui s'approche de celui proposé pour l'élimination des boucles dans la recherche d'un chemin qui vérifie la propriété de l'atteignabilité (en page 195 de la section 6.4.2 précédente).

Ainsi, nous avons besoin d'un prédicat `different(S1,S2,A)` (lignes 266–268) qui est vrai lorsqu'un automate `A` a des états locaux actifs différents dans les deux états globaux visités aux étapes `S1` et `S2`. Par contre, si `different(S1,S2)` n'est pas vrai, cela signifie que tous les états locaux actifs de tous les automates du AN sont les mêmes dans les deux états globaux. Et donc, nous pouvons déduire qu'il y a un cycle entre `S1` et `S2` (ligne 270) car il s'agit du même état global visité à ces deux étapes.

Nous éliminons finalement tous les chemins qui ne sont pas des cycles de longueur `n` avec la contrainte de la ligne 273. Cette dernière vérifie si les états globaux visités à l'étape initiale (c'est-à-dire à `S1=0`) et à l'étape finale (c'est-à-dire à `S2=n`) sont identiques.

```
264 % Les etats globaux des etapes S1 et S2 sont differents s'il y a au moins un automate
265 % ayant des etats locaux actifs differents dans ces deux etats globaux
266 different(S1,S2,A) ← active(level(A,I), S1), active(level(A,J), S2),
267         I!=J, step(S1), step(S2), automaton(A), S1!=S2.
268 different(S1,S2) ← different(S1,S2,_).
269 % Il existe un cycle entre S1 et S2 s'il n'y a pas de differences entre eux
270 cycle(S1,S2) ← not different(S1,S2), step(S1), step(S2), S1!=S2.
271 % Eliminer tout ensemble de reponses dans lequel il n'y a pas de cycle entre l'etat global
272 % initial et l'etat global final (i.e., la trace correspondante n'est pas une SCC)
273 ← not cycle(0, n).
```

Comme c'est montré au lemme 5.2 en page 149, si nous ne gardons que les chemins qui sont des cycles, leurs traces sont des composantes fortement connexes (SCC). Nous avons enfin besoin de vérifier si ces traces sont des domaines pièges (lemme 5.3 en page 150) afin de distinguer alors les attracteurs.

Énumération des attracteurs :

À cause du non-déterministe dans la dynamique, chaque état global dans le graphe d'états de transition d'un AN peut avoir plusieurs successeurs, et donc un chemin cyclique n'est

pas nécessairement un attracteur. En effet, il peut y avoir un état global qui appartient à la trace du cycle mais qui a d'autres successeurs qui sont différents de ceux de cette trace (voir le lemme 5.4 et son exemple en page 151). Une seule exception dans laquelle on peut dire que la trace de tout cycle est un attracteur est dans le cas d'une sémantique synchrone déterministe. Ceci se produit pour les modèles booléens, comme expliqué dans la section 5.2.3 en page 140. Dans ce cas (booléen et synchrone), le calcul peut s'arrêter ici parce qu'un cycle est nécessairement un attracteur (voir corollaire 5.2 en page 154). Ce résultat est utilisé dans (Dubrova & Teslenko, 2011) et dans (Qu, Yuan, Pang & Mizera, 2015) car ils cherchent des attracteurs dans les systèmes de régulation biologiques qui sont booléens et dont la dynamique suit une sémantique de mise à jour synchrone. En revanche dans notre travail, nous abordons un cas plus général et qui est plus difficile en étudiant une dynamique non-déterministe (synchrone et asynchrone).

Dans le cas général, certaines transitions peuvent permettre à la dynamique d'échapper au cycle. Dans un tel cas, la trace du cycle n'est pas un domaine piège (voir lemme 5.4 en page 151 et l'exemple qui suit ce lemme). C'est pourquoi une autre vérification est nécessaire pour filtrer tout cycle dont la trace peut être échappée par la dynamique (et qui n'est donc pas un attracteur). Encore une fois, dans un programme ASP ce filtrage est réalisé avec des règles qui sont des contraintes. En effet, réussir à définir les bonnes contraintes pour résoudre un problème avec un programme ASP est la démarche la plus appropriée. En plus, c'est plus efficace pour avoir des réponses en temps plus court.

Pour utiliser les contraintes, nous devons décrire le comportement que nous ne souhaitons pas observer : l'échappement de la trace du cycle. Pour cela, il est nécessaire de différencier entre les transitions locales effectivement activées (`played`) et les transitions locales jouables mais qui n'ont pas été activées (`alsoPlayable` à la ligne ligne 275). Ensuite, nous vérifions à chaque étape `S`, comprise entre `0` et `n`, si ces transitions qui sont également jouables, définies par le prédicat `"alsoPlayable"`, peuvent faire évoluer le AN ou non vers un nouvel état global qui ne fait pas partie de la trace du cycle trouvé.

Pour la sémantique asynchrone, toute transition locale qui est également jouable peut potentiellement faire en sorte que la dynamique quitte le cycle (rappelons qu'en asynchrone une seule transition locale est jouée à chaque étape). En ce qui concerne la sémantique synchrone, une transition locale également jouable doit nécessairement être en concurrence (voir définition 5.9 en page 141) avec une transition locale utilisée pour trouver le cycle étudié. Néanmoins, les deux cas sont abordés conjointement. Le prédicat `alsoPlayable(T,S)` indique qu'une transition locale `T` est également jouable dans l'état global de l'étape `S` dans le cycle considéré, mais n'a pas été utilisée pour construire spécifiquement le cycle. Ce prédicat est similaire au prédicat `played` utilisé précédemment dans les lignes 174 et 180.

```
274 % Calculer les transitions locales qui sont egalement jouables a l'etape S
275 alsoPlayable(T,S) ← not unPlayable(T,S),  not played(T,S), localTrans(T), step(S).
```

Après avoir trouvé les transitions locales qui sont également jouables dans chaque état global visité par le cycle, on doit vérifier si les états globaux, résultant de l'activation de ces transitions qui sont également jouables, font aussi partie de la trace du cycle ou pas. En effet, il est possible d'avoir d'autres transitions locales jouables qui font évoluer la dynamique du AN vers des états du cycle (c'est-à-dire vers des états globaux de la trace du cycle). Nous avons prouvé cette caractéristique dans le lemme 5.4 en 151 et après lequel nous avons donné un exemple pour mieux comprendre ces cas. Dans le programme ASP, ceci est vérifié par le prédicat `evolveInCycle` défini par la règle à la lignes 278–279.

Brièvement, on peut dire que de telles transitions locales sont simplement des "raccourcis" vers d'autres états globaux visités par le même cycle. C'est le cas des attracteurs complexes, qui ne se composent pas simplement d'un seul cycle mais de plusieurs cycles. De telles transitions globales ne sont pas prises en compte dans le cas présent comme nous sommes seulement intéressés à identifier les transitions globales qui permettraient à la dynamique du AN de s'échapper du cycle. Au lieu de cela, nous sommes intéressés à filtrer les cas où une transition permet de sortir du cycle (c'est-à-dire, conduit à un état global non présent dans la trace du cycle) en utilisant la contrainte de la ligne 282.

```
276 % La transition T est egalement jouable a l'etape S mais evolue le AN vers
277 % un etat global qui est deja visite a une etape Si (i.e., il appartient a la trace du cycle)
278 evolveInCycle(T,S,Si) ← alsoPlayable(T,S), target(T,B,K),
279         active(level(B,K),Si), Si!=S+1, step(Si), 1={different(S,Si,_)}.
280 % Eliminer les ensembles de reponses quand il y a une transition T qui est egalement jouable
281 % et qui fait evoluer le AN vers un etat global n'appartenant pas a la trace du cycle
282 ← alsoPlayable(T,S), not evolveInCycle(T,S,_).
```

Exemple. Dans le graphe d'états de la la figure 5.4 en page 146 du chapitre 5 précédent qui est calculé selon la sémantique asynchrone de mise à jour de la dynamique pour le AN de la figure 5.1 en page 134. Rappelons que ce AN est écrit dans un programme ASP dans les lignes 142–146 en page 187 (pour la définition de ses automates) et dans les lignes 148–162 en page 188 (pour la définition de ses transitions locales).

Soit le cycle suivant de longueur $n = 2$: $\langle a_1, b_2, c_0, d_1\rangle \rightarrow_{U^{\text{asyn}}} \langle a_1, b_2, c_1, d_0\rangle \rightarrow_{U^{\text{asyn}}} \langle a_1, b_2, c_0, d_1\rangle$. Dans la figure 5.4 en page 146 l'état global $\langle a_1, b_2, c_0, d_1\rangle$ est coloré en vert.

En suivant les programmes décrits ci-dessus dans cette section, l'un des ensembles de réponses qui pourrait permettre de trouver ce cycle, entre autres, il renvoie les prédicats suivants :

```
283 active(level("a",1),0) active(level("b",2),0)
284 active(level("c",0),0) active(level("d",1),0)
285 active(level("a",1),1) active(level("b",2),1)
286 active(level("c",0),1) active(level("d",0),1)
287 active(level("a",1),2) active(level("b",2),2)
288 active(level("c",0),2) active(level("d",1),2)
289 played(11,0) played(9,1)
290 alsoPlayable(1,0) alsoPlayable(6,0)
291 alsoPlayable(1,1) alsoPlayable(6,1)
292 cycle(0,2)
```

Les 3 états globaux visités dans les 3 étapes du cycle sont étiquetés `0`, `1` et `2`. Les états locaux actifs qu'ils contiennent sont décrits par le prédicat `active` dans les lignes 283–286. On note que les états globaux visités aux étapes `0` et `2` sont identiques. Ce qui est prouvé par le prédicat `cycle(0,2)` à la ligne 292. En outre, le prédicat `played` donne les deux transitions locales (étiquetées `9` and `11`, voir lignes 159 et 161 en page 188 où le AN est décrit en ASP) permettant de parcourir tous les états globaux du cycle. Tandis que le prédicat `alsoPlayable` donne les transitions locales qui sont également jouables dans des états globaux du cycle. En effet, dans les 2 états globaux visités par le cycle, aux étapes `0` et `1`, les transitions locales étiquetées par `1` et `6` sont jouables. Finalement, on voit que le prédicat `evolveInCycle` n'est jamais vrai pour cet exemple (il n'apparait pas dans l'ensemble de réponses). Ceci est dû au fait que les deux transitions locales qui sont également jouables font évoluer la dynamique du AN vers d'autres états globaux qui n'appartiennent pas à la trace de ce cycle. On peut visualiser ceci dans la figure 5.4 en page

146 où $\langle a_1, b_2, c_0, d_1\rangle$ (coloré en vert) a d'autres successeurs différents de $\langle a_1, b_2, c_0, d_0\rangle$ et aussi $\langle a_1, b_2, c_0, d_0\rangle$ a d'autres successeurs différents de $\langle a_1, b_2, c_0, d_1\rangle$. Par conséquent, cet ensemble de réponse est éliminé par la contrainte de la ligne 282 et il n'est pas affiché parmi les résultats.

Jusqu'à ce point, nous avons réussi à proposer un programme ASP qui énumère tous les attracteurs dans un AN donné et qui consiste en la trace d'un chemin de longueur `n`. Dans de nombreux cas, à l'exception de certains attracteurs complexes, cette longueur `n` est égale au nombre d'états globaux visités (i.e., la taille de la trace du chemin). C'est un cas trivial d'un chemin *minimal* couvrant un attracteur donné, c'est-à-dire qu'aucun chemin de longueur plus petite ne peut le couvrir. En effet, comme dans les exemples d'attracteurs dans les figures 5.4 et 5.5 en page 146, l'énumération de tous les chemins de longueur $n = 2$ est suffisante pour obtenir tous les attracteurs de taille 2, et il en va de même pour les attracteurs de taille 4. Mais sans la contrainte que nous développons ci-dessous (donnée dans les lignes 294–318), lorsque le programme est exécuté pour afficher les attracteurs couverts par un cycle de longueur n, il renvoie également tous les attracteurs de taille inférieure à n en considérant des cycles non minimaux ; c'est-à-dire le cycle de longueur n contient des cycles "non nécessaires" pour trouver tous les états globaux de l'attracteur, ou même des répétitions de tout le cycle. En l'occurrence, dans l'exemple des figures 5.4 et 5.5 en page 146, pour un $\mathtt{n} = 6$, jusqu'ici, le programme renvoie aussi tous les attracteurs de taille 2, parce que chacun d'eux peut être d'autant plus couvert par un cycle de longueur $n = 6$ qui est composé d'autres cycles de taille 2 répétés trois fois.

Par contre, dans le cas *des attracteurs complexes*, le problème est à l'opposé. Le cycle de longueur minimale couvrant tous les états globaux de l'attracteur complexe doit contenir des cycles "nécessaires" de plus petite longueur (i.e., revisiter des états globaux) afin de pouvoir couvrir tous les états globaux de l'attracteur. Pour plus de détails sur les attracteurs complexes voir page 154. Mais nous rappelons ici brièvement, l'exemple de « l'attracteur en étoile » représenté dans la figure 6.7 ci-dessous. Il est présenté dans un modèle comprenant les transitions globales suivantes : $\{s_0 \to s_1, s_1 \to s_0, s_1 \to s_2, s_1 \to s_3, s_2 \to s_1, s_3 \to s_1\}$. Le seul attracteur de ce modèle est l'ensemble $\mathcal{S} = \{s_0, s_1, s_2, s_3\}$. Nous remarquons qu'il n'est pas possible de couvrir tous les états globaux de cet attracteur sans visiter s_1 au moins 3 fois (même en ignorant l'étape finale qui est inévitablement répétée du cycle). En effet, un chemin possible pour le couvrir entièrement est : $s_0 \to s_1 \to s_2 \to s_1 \to s_3 \to s_1 \to s_0$ qui est de longueur `n`=6. Et aucun chemin d'une longueur inférieure n'existe pour couvrir cet attracteur bien que la taille de sa trace est égale à 4.

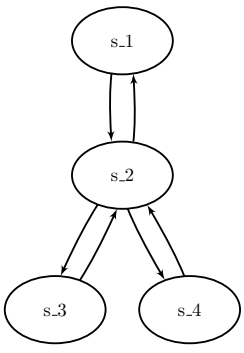


Figure 6.7 : Rappel de l'exemple d'un attracteur complexe appelé "attracteur en étoile".

Donc, l'objectif de la prochaine étape est d'exclure les cas où le cycle couvrant l'attracteur identifié n'est pas de longueur minimale.

n est la longueur minimale des cycles qui parcourent tous les états globaux de l'attracteur :

Nous rappelons que nous avons discuté davantage dans la section 5.4.4 en page 156, l'intérêt de ce raffinement du résultat. Et nous avons introduit cette méthode par l'algorithme 7 en page 159.

Le défi ici est de vérifier les deux cas de la minimalité dans le même programme ASP : (a) exclure tout ensemble de réponses comportant un attracteur qui peut être couvert par un cycle de longueur inférieure à *n* (b) et tout en retournant les attracteurs complexes pour lesquels le cycle couvrant tous leurs états globaux est strictement plus grand que la taille de leurs traces (c'est-à-dire *n*).

Pour cela, nous utilisons directement le résultat du lemme 5.1 en page 140 qui lie la longueur `n` d'un cycle (qui est aussi un chemin) à la taille de sa trace (`X`). Dans notre cas : `X = n + 1 - k`, où `k` est le nombre d'états globaux successivement répétés dans le cycle (voir définition 5.8 en page 140). Cette formule est implémentée dans les lignes 294–300 telle que la taille de la trace est trouvée par le prédicat `traceSize`. Ce dernier est également utilisé pour afficher à l'utilisateur la taille de l'attracteur trouvé et qui peut être strictement inférieur à la valeur de `n` (dans le cas des attracteurs complexes).

```
293 % L'etat global visite a l'etape S1 est aussi visite a l'etape S2 et S3 avec  S1 < S3 < S2
294 existSameCycleInside(S1,S2) ← cycle(S1,S3), S1<S3, S3<S2, step(S1), step(S2).
295 % Le premiere revisite de l'etat global de S1 a S2
296 repeatedState(S1,S2) ← cycle(S1,S2), not existSameCycleInside(S1,S2), S1<S2.
297 % K est le nombre des etats repetes tout au long de n etapes
298 getNbreRepeatedStates(K) ← K={repeatedState(_,_)}.
299 % X est la taille de la trace du cycle de longueur n
300 traceSize(X) ← getNbreRepeatedStates(K), X=n+1-K.
```

Notre objectif dans ce qui suit est de proposer un programme qui identifie autant que possible, tous les attracteurs du AN qui sont atteints par un cycle de longueur `n` et qui est minimale. Nous proposons les règles des lignes 305–316 ci-dessous, pour vérifier cette propriété ; chacune d'elles conclut avec le prédicat `isNotMinimal(n)`, ce qui signifie que le cycle considéré n'est pas minimal pour l'attracteur trouvé. En fin du programme, `isNotMinimal(n)` est utilisé dans la contrainte de la ligne 318 qui élimine toutes ces réponses indésirables du résultat.

Nous vérifions d'abord s'il existe un chemin de longueur `X <n` sans qu'il y ait des répétitions d'états entre l'étape `0` à l'étape `X`, où `X` est la taille de la trace du cycle (c'est-à-dire `X` est le nombre d'états globaux distinct visités au court du cycle). Ensuite, nous vérifions également s'il y a une transition de l'état global à l'étape `X` vers l'état global de l'étape `0`. Si les deux propriétés sont vraies, alors il existe un chemin de longueur `X<n` qui couvre tous les états globaux de l'attracteur. Et donc `n` n'est pas la longueur minimale qui peut couvrir tous les états globaux de cet attracteur (lignes 305–308).

Un autre cas qui peut se produire tel que `n` est non minimal est détaillé dans les lignes 311–312. C'est lorsqu'il existe des raccourcis entre certains états globaux du cycle. En outre, un chemin de longueur minimale ne permet pas des répétitions entre les états globaux successifs à l'intérieur d'un cycle (ligne 314). Enfin, lorsqu'un cycle entier est

répété plusieurs fois, alors le nombre de répétitions est évidemment supérieur au maximum prévu c'est-à-dire à `n` (ligne 316). Comme indiqué précédemment, dans n'importe lequel des cas précédents, le cycle considéré n'est pas de longueur minimale pour parcourir tous les états globaux de l'attracteur, et donc la réponse est rejetée par la contrainte à la ligne 318.

```
301 % Il existe d'autres cycles entre S1 et S2
302 existCyclesInside(S1,S2) ← cycle(S,Sbis), S1<=S, S<=S2, S1<=Sbis, Sbis<=S2,
303         S!=Sbis, step(S1;S2).
304 % Il existe un chemin plus court de longueur X visitant tous les etats globaux
305 isNotMinimal(n) ← evolveInCycle(T,S,0), not existCyclesInside(0,S), S=X-1,
306         traceSize(X), X<n.
307 isNotMinimal(n) ← evolvedInCycle(T,0,S), S=X-1, not existCyclesInside(S,0),
308         traceSize(X), X<n.
309 % L'attracteur n'est pas complexe et tous ses etats globaux peuvent etre atteints
310 % par un cycle de longueur inferieure a n
311 isNotMinimal(n) ← evolveInCycle(T,S1,S2), not cycle(S1,_), not cycle(S2,_),
312         traceSize(X), X<n.
313 % Pas de cycles inutiles entre les etapes successives
314 isNotMinimal(n) ← cycle(S1,S2), cycle(S1+1,S2+1).
315 % Pas de cycles inutiles dans le cycle de longueur n
316 isNotMinimal(n) ← getNbreRepeatedStates(Y), traceSize(X), Y>X.
317 % n n'est pas la longueur minimale pour atteindre tous les etats globaux de cet attracteur
318 ← isNotMinimal(n).
```

Nous notons que ces contraintes sont pertinentes pour la dynamique non-déterministe, quelle que soit sa sémantique : asynchrone ou synchrone.

Néanmoins, il existe encore un cas de répétitions qui ne peut pas être résolu par la contrainte précédente (ligne 318) : l'existence de plusieurs cycles minimaux pour le même attracteur mais tels que ces cycles sont différents. En effet, pour un attracteur donné, il est possible de trouver plusieurs cycles minimaux qui couvrent tous ses états globaux en changeant l'état global initial, ou le sens du parcours (e.g., dans le cas des attracteurs complexes). Par exemple, l'attracteur hypothétique $\{\zeta_0\,;\zeta_1\}$ est capturé par deux cycles : $\zeta_0 \to \zeta_1 \to \zeta_0$ et $\zeta_1 \to \zeta_0 \to \zeta_1$. Le résultat final à l'intérieur de chaque ensemble de réponses est décrit par les atomes `active(Lv,S)`, où `S` indique l'étiquette d'une des étapes du cycle, et `Lv` correspond à l'un des états locaux actifs (e.g., `level(A,I)`).

Dans cette section, nous avons donné un programme ASP pour énumérer tous les attracteurs cycliques de taille inférieure ou égale à $\mathtt{n} \in \mathbb{N}^*$ dans un AN donné. Rappelons qu'un attracteur cyclique est un ensemble d'états globaux de taille supérieure ou égale à 2. Et dans la section suivante, nous cherchons à identifier les attracteurs singletons appelés des points fixes.

6.5.2 Les points fixes en ASP

L'énumération des points fixes nécessite la traduction de la définition d'un point fixe (donnée dans la définition 5.14 en page 5.14) en règles logiques suivant la syntaxe d'ASP. Nous rappelons que l'algorithme 8 en page 160 donne l'idée intuitive du programme ASP qui énumère les points fixes dans un AN, c'est-à-dire pour lesquels aucune transition n'est jouable.

Ainsi, la première étape consiste à énumérer tous les états globaux possibles du AN. En d'autres termes, toutes les combinaisons possibles d'états locaux sont générées en choisissant exactement un état local de chaque automate.

Cependant, avant de calculer ces combinaisons, nous voulons prétraiter la liste des états locaux afin d'exclure chaque état local a_i qui s'auto-active ou s'auto-inhibe ; c'est-à-dire il existe une transition locale $a_i \xrightarrow{\emptyset} a_j$ dans l'ensemble des transitions locales du AN. En effet, ces états locaux ne peuvent pas être stables, parce qu'une telle transition locale est toujours jouable. Ceci est fait par les lignes 320–324.

```
319 % La transition T n'est pas une auto-transition
320 notSelfTransition(T) ← condition(T,B,_), target(T,A,_), A!=B.
321 % Cacher les etats locaux des automates ayant une auto-transition
322 hiddenAutomatonLevel(A,I) ← not notSelfTransition(T), condition(T,A,I).
323 % Les etats locaux qui ne sont pas caches
324 shownAutomatonLevel(A,I) ← not hiddenAutomatonLevel(A,I), automatonLevel(A,I).
325 % Trouver tous les etats globaux possibles
326 1 { fix(A,I) : shownAutomatonLevel(A,I) } 1 ← automaton(A).
327 % T est une transition qui n'est pas jouable dans l'etat global considere
328 unPlayable(T) ← fix(A,I), condition(T,A,J), I!=J.
329 % L'etat global est elimine s'il a des transitions locales jouables
330 ← not unPlayable(T), transition(T).
```

La ligne 326 présente une règle avec cardinalité (c'est-à-dire avec les accolades dans la tête de la règle) telle que définie à la ligne 6.6 en page 167 dans la section 6.2.1. Elle énumère tous les états globaux à prendre en compte en créant autant d'ensembles de réponses. L'état global est alors défini en considérant pour chaque automate, `automaton`, exactement un état local parmi ceux qui ne sont pas cachés (car ils ont une auto-transition définies à la ligne 320) Ainsi, l'état global est sélectionné parmi les états locaux définis par le prédicat `shownAutomatonLevel` (ligne 324). Nous déclarons chaque état global par un ensemble de prédicats `fix(A,I)` (appelés ainsi en prévision des résultats finaux pour l'identification des points fixes) où `I` est l'état local actif de l'automate `A`.

La dernière étape consiste à filtrer n'importe quel état global ζ, parmi tous ceux qui sont générés, qui n'est pas un point fixe. Dans ce cas, il consiste à éliminer tous les ensembles de réponses dans lesquels l'état global correspondant a au moins une transition locale qui est jouable (donc $\mathsf{J}(\zeta) \neq \emptyset$ avec $\mathsf{J}(\zeta)$ est l'ensemble des transitions locale jouables dans ζ). Un tel filtrage est idéalement réalisé avec l'utilisation d'une ou de plusieurs contraintes. Et, comme nous l'avons indiqué précédemment, une contrainte en ASP supprime toute réponse qui satisfait sa partie à droite.

En ce qui concerne notre problème, un ensemble de réponses est éliminé s'il existe au moins une transition locale jouable dans l'état global considéré (ligne 330). Une transition `T` est considérée comme non jouable (c'est-à-dire $\mathtt{T} \notin \mathsf{J}(\zeta)$), si au moins une de ses conditions n'est pas satisfaite. Pour cela, le prédicat `unPlayable(T)` défini à la ligne 328, identifie les transitions locales non jouables quand l'une de ses conditions (`condition`) a un état local différent de celui qui est actif dans l'état global considéré. Ceci est utilisé dans la dernière contrainte (ligne 330) qui dit que s'il existe une transition locale qui est jouable dans l'état global considéré (i.e., $\exists\, \mathtt{T} \in \mathcal{T}$, $\mathtt{T} \notin \mathsf{J}(\zeta)$), alors cet état global doit être éliminé du résultat (parce qu'il n'est pas un point fixe). Finalement, les points fixes du AN considéré sont exactement les états globaux représentés dans chaque ensemble de réponses restant, décrit par les atomes `fix(A,I)` qui définit l'état local actif de chaque automate.

Exemple. Le modèle AN de la figure 5.1 en page 134 contient 4 automates : a et c ont 2 états locaux tandis que b et d en ont 3. Par conséquent, le modèle AN entier a $2 \times 2 \times 3 \times 3 = 36$ états globaux (qui sont atteignables ou pas depuis un état global

initial donné). Nous pouvons vérifier que ce modèle contient exactement 3 points fixes : $\langle a_1, b_1, c_0, d_0\rangle$, $\langle a_1, b_1, c_1, d_0\rangle$ et $\langle a_0, b_0, c_0, d_1\rangle$. Tous les trois sont représentés et colorés en rouge dans les deux figures 5.4 et 5.5 en page 146. Dans ce modèle, aucun autre état global ne vérifie cette propriété. Nous rappelons que les points fixes sont identiques pour les sémantique synchrone et asynchrone.

Si nous exécutons le programme ASP détaillé ci-dessus (lignes 320–330) avec la description du modèle AN donné dans l'exemple 6.4.1 en page 187 (lignes 142–162), 3 ensembles de réponses correspondent au résultat attendu. La sortie de Clingo est la suivante :

```
Answer: 1
fix("a",0) fix("b",1) fix("c",0) fix("d",1)
Answer: 2
fix("a",0) fix("b",0) fix("c",0) fix("d",1)
Answer: 3
fix("a",1) fix("b",1) fix("c",0) fix("d",0)
```

Le problème de trouver des attracteurs dans un réseau discret est NP-complet, donc la mise en œuvre que nous avons donnée dans cette section est également d'une telle complexité. Cependant, les solveurs ASP (à savoir, Clingo dans notre cas) sont spécialisés dans la résolution de problèmes aussi complexes. La section suivante est consacrée aux résultats de plusieurs expériences de calcul que nous avons effectuées sur des réseaux biologiques afin de montrer que notre implémentation ASP peut gérer de grands systèmes et renvoyer le résultat en seulement quelques secondes dans plusieurs cas.

6.5.3 Applications

Dans cette section, nous présentons plusieurs expériences menées sur des réseaux biologiques. Nous détaillons d'abord les résultats de nos programmes sur l'exemple du modèle AN de la figure 5.1 en page 134. Ensuite, nous résumons les résultats et la performance de certaines expériences effectuées sur d'autres modèles dont le nombre de composants peut aller jusqu'à 100. En général, les performances temporelles sont bonnes et les résultats globaux semblent appuyer notre utilisation de l'ASP pour la vérification des propriétés formelles (logiques) ou l'énumération des constructions spéciales sur les systèmes biologiques.

Toutes les expériences ont été exécutées sur un ordinateur de bureau avec un processeur Pentium VII 3 GHz et 16 Go de RAM.

6.5.3.1 Un premier cas d'étude simple

Nous avons d'abord effectué des expériences détaillées sur le modèle AN à 4 composants de la figure 5.1 en page 134. Comme indiqué précédemment, ce réseau contient 4 automates et 12 transitions locales. Le graphe d'états de transitions comprend 36 états globaux différents et certains d'entre eux sont détaillés dans les graphes d'états de la figure 5.4 en page 146 (pour la sémantique asynchrone) et de la figure 5.5 en page 147 (pour la sémantique synchrone).

L'étude analytique des domaines pièges minimaux sur ce petit modèle permet de trouver les attracteurs et les points fixes attendus en fonction de la sémantique de mise à jour de la dynamique. Dans la suite, nous assimilons les points fixes aux attracteurs de longueur $\mathtt{n} = 0$ et qu'ils ont alors une trace de taille 1 :

– Sémantique asynchrone :

- $\mathtt{n} = 0$: $\langle a_1, b_1, c_1, d_0\rangle$,$\langle a_1, b_1, c_0, d_0\rangle$ et $\langle a_0, b_0, c_0, d_1\rangle$;
- $\mathtt{n} = 2$: $\{\langle a_0, b_1, c_0, d_0\rangle, \langle a_0, b_1, c_0, d_2\rangle\}$;
- $\mathtt{n} = 4$: $\{\langle a_1, b_2, c_1, d_1\rangle, \langle a_0, b_2, c_1, d_1\rangle, \langle a_0, b_2, c_1, d_0\rangle, \langle a_1, b_2, c_1, d_0\rangle\}$.

– Sémantique synchrone :

- $\mathtt{n} = 0$: $\langle a_1, b_1, c_1, d_0\rangle$,$\langle a_1, b_1, c_0, d_0\rangle$ et $\langle a_0, b_0, c_0, d_1\rangle$;
- $\mathtt{n} = 2$: $\{\langle a_0, b_1, c_0, d_0\rangle, \langle a_0, b_1, c_0, d_2\rangle\}$ and $\{\langle a_1, b_2, c_1, d_1\rangle, \langle a_0, b_2, c_1, d_0\rangle\}$.

Les points fixes ($\mathtt{n} = 0$) sont trouvés par la méthode de la section 6.5.2 précédente et les attracteurs cycliques ($\mathtt{n} > 1$) sont trouvés par la méthode de la section d'après (section 6.5.1 en page 200).

Lorsqu'ils sont donnés à un solveur, les programmes ASP introduits dans les sections précédentes, produisent directement les solutions attendues. Le résultat pour l'énumération des points fixes est donné dans l'exemple précédent en page 207. Le résultat pour l'énumération des attracteurs cycliques est donné ci-dessous pour les deux sémantiques. Nous notons que chaque état global appartenant à un attracteur est étiqueté avec un nombre (par exemple, `0` et `1` pour les cas où $\mathtt{n} = 2$) de sorte que chaque état local actif est présenté par un atome indépendant.

```
--- Asynchronous n=2: ---
Answer: 1
active(level("a",0),0) active(level("b",1),0) active(level("c",0),0)
active(level("d",0),0) active(level("a",0),1) active(level("b",1),1)
active(level("c",0),1) active(level("d",2),1) traceSize(2) cycle(0,2)

--- Asynchronous n=4: ---
Answer: 1
active(level("a",1),0) active(level("b",2),0) active(level("c",1),0)
active(level("d",1),0) active(level("a",0),1) active(level("b",2),1)
active(level("c",1),1) active(level("d",1),1) active(level("a",0),2)
active(level("b",2),2) active(level("c",1),2) active(level("d",0),2)
active(level("a",0),3) active(level("b",0),3) active(level("c",0),3)
active(level("d",1),3) traceSize(4) cycle(0,4)

--- Synchronous n=2: ---
Answer: 1
active(level("a",0),0) active(level("b",1),0) active(level("c",0),0)
active(level("d",0),0) active(level("a",0),1) active(level("b",1),1)
active(level("c",0),1) active(level("d",2),1) traceSize(2) cycle(0,2)
Answer: 2
active(level("a",1),0) active(level("b",2),0) active(level("c",1),0)
active(level("d",1),0) active(level("a",0),1) active(level("b",2),1)
active(level("c",1),1) active(level("d",0),1) traceSize(2) cycle(0,2)
```

En outre, l'exécution des programmes ASP avec $\mathtt{n} \neq 2$ et $\mathtt{n} \neq 4$ ne renvoie aucun résultat. Ce qui signifie que le solveur termine correctement, en renvoyant `unsatisfaible`, après n'avoir trouvé aucune réponse. Ceci est attendu, car il n'y a pas d'attracteur de longueur différente de 2 et de 4 pour ce modèle AN. En plus, nous avons exclu les cycles répétés des résultats ; donc les attracteurs qui sont trouvés de taille 2 et ceux de taille 4 ne sont pas trouvés pour $\mathtt{n} = 6$ ni pour un plus grand $\mathtt{n}$. Sur ce petit réseau, tous les résultats sont calculés en moins de 0,05 seconde.

6.5.3.2 Expériences

Dans ce qui suit, nous proposons des expériences supplémentaires pour montrer les capacités de nos programmes ASP pour des systèmes réels de taille importante. Nous ne donnons pas les détails des résultats de ces expériences mais nous nous concentrons plutôt sur la taille des attracteurs des sorties et sur le temps de calcul. Nous avons utilisé plusieurs réseaux booléens ou multi-valués préexistants modélisant des RRB réels trouvés dans la littérature. Nous donnons dans le tableau 6.7 ci-dessous, une description des caractéristiques (nombre de composants, de transitions, d'états globaux...) des modèles AN utilisés dans les expériences. Et nous décrivons avant ci-dessous les systèmes biologiques que chacun d'eux représente.

- **Lambda phage :** un réseau de régulation mettant en avant certains gènes viraux cruciaux dans la décision entre la lyse et la lysogénisation dans les régions tempérées du bacteriophage lambda (Thieffry & Thomas, 1995) ;
- **Trp-reg :** modèle qualitatif des voies métaboliques régulées de la biosynthèse du tryptophane dans le E. coli (Simao et al., 2005) ;
- **Fission-yeast :** Schizosaccharomyces Pombe, modèle de cycle cellulaire (Davidich & Bornholdt, 2008) ;
- **Mamm :** cycle cellulaire des mammifères (Fauré et al., 2006) ;
- **Tcrsig :** modèle de signalisation et de régulation de la voie de signalisation du TCR dans la différenciation des mammifères (Klamt et al., 2006) ;
- **FGF :** voie de signalisation du Drosophila FGF (Mbodj, Junion, Brun, Furlong & Thieffry, 2013) ;
- **T-helper :** modèle de la différenciation des cellules T-helper et de la plasticité, ce qui explique les nouveaux sous-types cellulaires (Abou-Jaoudé et al., 2014).

Pour obtenir ces modèles, nous les avons d'abord extraits du répertoire des modèles GINsim[15] (Chaouiya et al., 2012), dans le format GINML. Ces modèles correspondent aux réseaux asynchrones discrets donnés dans les articles scientifiques correspondants. Ensuite, l'étape de conversion vers un programme ASP est automatisée à l'aide des outils suivants :

- L'outil GINsim existant permet d'exporter ses modèles du format GINML vers le formalisme *SBML-qual* ;
- La bibliothèque LogicalModel[16] (Naldi, Monteiro, Müssel, Kestler, Thieffry, Xenarios, Saez-Rodriguez, Helikar, Chaouiya et al., 2015) qui peut convertir des modèles *SBML-qual* en modèles dans le formalisme des AN ;
- Enfin, nous avons développé un script pour convertir des modèles décrits dans le formalisme des AN en des programmes ASP tels que chaque programme suit les règles détaillées dans la section 6.4.1 en page 187.

Il est à noter que chacune de ces étapes de conversion préserve entièrement la dynamique entre les modèles concernant la sémantique asynchrone (Chatain, Haar, Jezequel, Paulevé & Schwoon, 2014). Ainsi, le programme ASP final est bisimilaire au modèle original au format GINML. Les caractéristiques de chaque modèle, une fois traduites en AN, sont

[15] `http://ginsim.org/models_repository`
[16] https ://github.com/colomoto/bioLQM

données dans le tableau 6.7 ci-dessous. Les résultats de nos expériences[17] sont donnés dans les tableaux 6.8 et 6.9 ci-dessous.

Modèles	**Description des modèles**			
	$\lvert\Sigma\rvert$	$\max_{a\in\Sigma}\{\lvert\mathcal{S}_a\rvert\}$	$\lvert\mathcal{T}\rvert$	$\lvert\mathcal{S}\rvert$
Exemple	4	3	12	36
Lambda phage	4	4	46	48
Trp-reg	4	3	14	36
Fission-yeast	9	3	43	$3 \times 2^9 = 1{,}536$
Mamm.	10	2	34	$2^{10} = 1{,}024$
Tcrsig	40	2	85	$2^{40} \simeq 10^{12}$
FGF	59	3	102	$2^{31} \simeq 1.2 \times 10^{10}$
T-helper	101	3	316	$2^{102} \simeq 5.7 \times 10^{31}$

Table 6.7 : Brève description des modèles utilisés dans nos benchmarks. Les lignes successives résument les informations relatives aux modèles respectivement de l'exemple cas d'étude de la figure 3.2, du bacteriophage lambda, de la regression de la biosynthèse du tryptophane dans le E.coli, du fission yeast, du cycle cellelaire du mammifère, de la voie de signalisation du FGF de la drosophile, la voie de signalisation du TCR dans la différenciation des mammifères, et la différenciation des cellules T-helper, Pour chacun d'eux, le tableau donne le nombre d'automates ($|\Sigma|$), le niveau local maximal dans les automates ($\max_{a\in\Sigma}\{|\mathcal{S}_a|\}$) le nombre de transitions locales ($|\mathcal{T}|$) et le nombre d'états dans le graphe d'états de transition correspondant ($|\mathcal{S}|$).

Modèles	**Énumération des points fixes pour les deux sémantiques**	
	Δt (ms)	**#PF**
Exemple	2	3
Lambda phage	4	1
Trp-reg	6	2
Fission-yeast	5	1
Mamm.	3	1
Tcrsig	5	8
FGF	25	1.536
T-helper	170.642	5.875.504

Table 6.8 : Résultats de nos énumérations de points fixes. Les lignes successives résument les informations concernant les modèles détaillés dans le tableau 6.7. Pour chaque modèle, le tableau indique le temps de calcul (Δt) pour l'énumération de tous les points fixes du modèle (leur nombre total est **#PF**).

Nous notons que tous les résultats pour la recherche de points fixes sont comparés et confirmés en utilisant GINsim (Chaouiya et al., 2012) et Pint (Paulevé, 2016*b*). En ce qui concerne les programmes pour l'énumération des attracteurs, nous avons comparé nos résultats avec l'outil BNS (Boolean Network System) de (Dubrova & Teslenko, 2011) pour la sémantique synchrone sur les modèles : **Fission-yeast**, **Mamm.** et **Tcrsig**, et pour la sémantique asynchrone les résultats obtenus sont comparés avec ceux de GINsim (Chaouiya

[17] Tous les programmes et les benchmarks sont disponibles à l'adresse suivante : `http://www.irccyn.ec-nantes.fr/~benabdal/attractors.zip`

et al., 2012) sur les modèles : **Lambda phage**, **Trp-reg**, **Fission-yeast** et **Mamm.**. Dans tous les cas, nous avons trouvé les mêmes résultats.

Il est intéressant de noter que notre méthode permet de retourner une réponse pour identifier les attracteurs de petite taille, même pour les grands modèles (100 composants). En revanche, d'autres outils (cités ci-dessous) peuvent prendre beaucoup de temps ou même ne pas répondre. Ce qui arrive avec GINsim pour le modèle **Tcrsig** et les modèles de tailles plus grandes (à partir de 40 composants). En effet, parce qu'elles sont basées sur le calcul du graphe d'états de transitions complet même pour l'identification des petits attracteurs.

Nos résultats n'ont pas pu être comparés avec, par exemple, la méthode ASP-G (Mushthofa et al., 2014). En effet, avec cet outil, l'utilisateur doit choisir une règle de mise à jour de la dynamique sur laquelle la simulation est basée. Par exemple, une règle qui consiste à activer un gène lorsqu'au moins l'un de ses activateurs est actif et qu'aucun de ses inhibiteurs n'est actif. Une autre règle consiste à activer un gène lorsqu'il a plus d'activateurs exprimés que des inhibiteurs actifs. Puisque la règle d'activation choisie est appliquée à tous les composantes du modèle, alors que les règles d'évolution de notre sémantique de la dynamique d'un AN sont spécifiques à chaque composant (c'est-à-dire des transitions locales pour chaque automate). Ainsi, les résultats des deux outils ne peuvent pas être strictement comparés.

Parmi les résultats obtenus, certains attracteurs peuvent être énumérés plusieurs fois dans les ensembles de réponses, malgré tout filtrage, comme expliqué à la fin de la section 6.5.1. En effet, le solveur renvoie des ensembles de réponses différents pour des chemins différents qui couvrent la même trace (c'est-à-dire le même ensemble d'états globaux) mais qui diffèrent entre eux par l'état global initial. Autrement dit, pour un même attracteur, nous affichons plusieurs chemins qui le parcourent en entier tels que l'état global initial de chaque chemin est différent de l'autre. C'est pourquoi, dans les résultats du tableau 6.9 ci-dessus, nous nous sommes concentrés sur les temps de la réponse et du calcul du tout premier attracteur trouvé de longueur `n`.

Dans le cas où l'utilisateur a besoin de la liste exhaustive de tous les attracteurs, notre méthode peut également lister dans sa sortie toutes les réponses, y compris celles répétées. Par exemple, elle retourne 4 réponses pour le modèle **Trp-reg** et dans chaque réponse on trouve un cycle de longueur 4 calculé selon la sémantique asynchrone, et le calcul prend 47 millisecondes. Cela représente typiquement un attracteur de taille 4 où chaque ensemble de réponses représente un cycle ayant des différents états initiaux. En ce qui concerne le modèle **T-helper** (le plus grand modèle étudié avec 101 automates), la recherche de tous ses attracteurs de taille `n` = 2 selon la sémantique synchrone prend environ 275 secondes ($\sim$ 5 minutes) et renvoie 2.058.272 réponses . En revanche, il prend seulement 57 secondes pour retourner tous les attracteurs de taille `n`=12 et renvoie 6.144 réponses. Cependant, comme expliqué précédemment, ces résultats indiquent que ce modèle a un nombre total d'attracteurs de taille `n`=12 qui est strictement inférieur à 6.144, car chaque attracteur est répété plusieurs fois dans ces 6.144 réponses.

Ainsi, afin de filtrer les répétitions restantes, il devrait être possible d'utiliser un script ou un éditeur de texte afin d'extraire uniquement les états globaux de chaque ensemble de réponses et de rejeter ainsi les réponses présentant exactement le même attracteur. Un tel traitement n'est pas trivial en ASP et présente l'une des cibles des travaux futurs.

Modèles	n	Énumération des attracteurs — Sémantique asynchrone Δt (ms)	∃?**A**	Sémantique synchrone Δt (ms)	∃?**A**
Exemple	2	7	**yes**	7	**yes**
	4	16	**yes**	14	no
	8	98	no	75	no
Lambda phage	2	14	**yes**	14	**yes**
	10	1,352	no	842	no
	20	15,656	no	14,452	no
Trp-reg	2	8	no	7	no
	4	14	**yes**	15	no
	20	3,908	no	3,808	no
Fission-yeast	2	16	no	16	**yes**
	10	1,011	no	807	no
	20	17,302	no	16,313	no
Mamm.	2	12	no	12	no
	7	177	no	147	**yes**
	10	720	no	605	no
	20	58,133	no	9,253	no
Tcrsig	2	26	no	25	no
	6	353	no	288	**yes**
	10	2,420	no	1,841	no
	20	85,599	no	27,078	no
FGF	2	38	no	36	no
	10	2,080	no	1,953	no
	20	30,861	no	29,838	no
T-helper	2	180	no	125	**yes**
	3	391	no	301	**yes**
	4	782	no	1,064	no
	6	4,271	no	2,372	**yes**
	7	7,909	no	3,522	**yes**
	9	26,443	no	7,042	**yes**
	10	44,924	no	12,208	**yes**
	12	107,358	no	28,520	**yes**
	20	4,230,836 $\sim$ 1h17	no	187,105 $\sim$ 3min	no

Table 6.9 : Résultats du programme ASP pour l'énumération des attracteurs. Les lignes successives résument les informations concernant les modèles détaillés dans le tableau 6.7 ci-dessus. Pour chaque modèle et pour les deux sémantiques (asynchrone et synchrone), le tableau montre, en fonction de la longueur du chemin donné (i.e., n) le temps de calcul pour le premier attracteur trouvé par le solveur (Δt), et une indication de l'existence ou pas d'un attracteur (∃?**A**).

6.5.4 Discussion

Nous envisageons d'étendre ce travail par des adaptations et des optimisations de l'approche pour aborder des modèles encore plus grands. Tout d'abord, l'option "projection" de Clingo qui affiche une seule réponse lorsque plusieurs ensembles de réponses contiennent des prédicats communs, est actuellement étudiée afin de filtrer les attracteurs répétés (i.e., qui apparaissent actuellement plusieurs fois parce qu'ils sont couverts par plusieurs cycles

possibles). Une autre piste consiste à retourner des approximations des résultats, c'est-à-dire parfois "enlever" certaines réponses, mais avec l'avantage d'une performance très améliorée. Encore une fois, l'application de divers filtres aux résultats générés peut éviter la redondance dans les résultats et guider le processus de la résolution du problème. Inversement, il peut être possible de réduire l'aspect incrémental du processus d'analyse, par exemple en recherchant des cycles de taille inférieure à (et non seulement égale à) une valeur `n` donnée, de sorte que l'utilisateur puisse directement commencer avec des valeurs plus élevées.

Bien entendu, d'autres extensions permettant d'aborder d'autres problèmes proches seraient intéressantes. Par exemple, le problème inverse de l'attracteur consiste à construire ou à énumérer des réseaux possédant un ensemble donné de propriétés d'attracteur, afin de répondre à des questions d'inférence de modèles. Nous aimerions également étendre ces méthodes basées sur ASP pour étudier d'autres propriétés intéressantes de modèles dynamiques telles que l'énumération des bassins d'attraction, des jardins d'Eden ou des bifurcations (Fippo-Fitime, Roux, Guziolowski & Paulevé, 2016).

6.6 Conclusion

Dans ce chapitre, nous avons présenté une nouvelle approche logique implémentée en ASP pour (i) apprendre des T-AN (ii) analyser dynamiquement la propriété d'atteignabilité dans les AN et dans les T-AN et (iii) identifier efficacement la liste des attracteurs dans les AN.

Nous avons formalisé nos approches en utilisant le formalisme des AN, qui est bisimilaire à de nombreux réseaux logiques (Chatain et al., 2014). Tous les résultats donnés ici peuvent donc être adaptés pour des formalismes plus généraux tels que le modèle de Thomas (Thomas, 1973) pour la sémantique asynchrone ou des modèles suivant la sémantique synchrone (Kauffman, 1969). En outre, ce cadre peut englober plusieurs sémantiques de la dynamique.

L'originalité de notre travail consiste en l'énumération de tous les modèles (pour l'apprentissage), de tous les chemins (pour la vérification de l'atteignabilité) et de tous les ensembles d'états globaux (pour l'identification des attracteurs) grâce à l'utilisation d'ASP (qui est un puissant paradigme de programmation déclarative). Le cadre computationnel est basé sur le formalisme des AN présumant une dynamique non-déterministe. Grâce aux codages que nous avons introduits et aux puissantes heuristiques développées dans les solveurs modernes pour ASP, nous sommes en mesure d'aborder ce type de problèmes complexes (NP-complet). Le principal avantage d'une telle méthode est d'obtenir une énumération exhaustive de tous les états globaux potentiels tout en étant encore praticable pour les modèles avec une centaine de composants en interaction. Comme l'identification des attracteurs peut donner un aperçu du comportement à long terme des systèmes biologiques, aborder cette question est un défi auquel nous avons contribué. En outre, nous espérons que notre travail aidera à ouvrir de nouvelles voies et outils pour explorer ce domaine qui reste riche de multiples autres perspectives intéressantes.

Ainsi, nous achevons ce manuscrit par une conclusion générale et quelques perspectives dans le chapitre 7 suivant.

Chapitre 7

Conclusion et perspectives

Nous revenons dans ce chapitre sur les apports de la présente thèse. Nous commençons par un récapitulatif de tout ce que nous avons mis en place au fil des chapitres. (i) nous enrichissons la dynamique des réseaux d'automates par l'intégration d'une composante temporelle et de ce qu'elle concerne précisément ; (ii) nous présentons l'apprentissage de tels réseaux modélisant des systèmes biologiques ; (iii) nous étudions ces systèmes concurrents par l'analyse et la vérification des propriétés dynamiques (identification des attracteurs et vérification de la propriété d'atteignabilité). Enfin, nous évoquons plusieurs nouvelles perspectives scientifiques en prolongement de nos contributions.

7.1 Introduction

Dans le but d'étudier les systèmes biologiques, plusieurs recherches ont été menées pour étudier la dynamique des réseaux de régulation biologique (RRB) qui les modélisent. Ainsi, plusieurs formalismes ont été introduits pour représenter les RRB allant des systèmes d'équations différentielles à la programmation logique, en passant par des modèles probabilistes.

L'utilisation de **modèles discrets** présente une abstraction puissante de la complexité inhérente à ces systèmes, tout en conservant certaines propriétés dynamiques intéressantes. L'utilisation de tels modèles discrets est initiée par Stuart A. Kauffman (1969), plus tard suivi par René Thomas (1973) qui y ajoute une dynamique asynchrone et par El Houssine Snoussi (1989) qui propose la notion de paramètres discrets pour définir totalement la dynamique. Ces formalismes de modèles discrets ont été massivement étudiés, comme nous l'avons évoqué au chapitre 2 (à la section 2.2.2 en page 27).

De plus, certains paramètres (priorité, temps, probabilité...) peuvent être très intéressants quand ils sont intégrés dans le modèle. En effet, ils expriment un raffinement de la dynamique du modèle pour l'approcher de la dynamique réelle du système modélisé. Dans cette thèse nous nous sommes intéressés principalement aux **paramètres temporels**.

En effet, l'émergence d'une large gamme de nouvelles technologies a permis de produire une quantité massive de données biologiques. Ainsi, une grande quantité de **données de séries temporelles** est maintenant produite. Elles décrivent l'évolution des concentrations des composants du système biologique au cours du temps.

Ces données de séries temporelles offrent donc la possibilité de prendre en compte la notion de **chronométrie** dans le processus de modélisation. Cet ajout permet de proposer des modèles hybrides sur lesquels il est possible d'étudier des nouvelles propriétés dynamiques beaucoup plus précises telles que les dynamiques quantitatives.

Dans cette thèse, nous avons apporté une contribution à la modélisation des systèmes concurrents avec l'intégration d'une composante temporelle (appelée le **délai**). Celle-ci est spécifique à chaque interaction dans le réseau et il représente le temps nécessaire pour que cette interaction s'exécute. Nous sommes passés alors du formalisme des **réseaux d'automates** (*Automata Networks*, AN) au formalisme des **réseaux d'automates avec le temps** (*Timed Automata Networks*, T-AN) qui présente une dynamique plus raffinée. Dans ce but, nous avons proposé une nouvelle sémantique appropriée de la dynamique des T-AN.

En nous basant sur cette nouvelle sémantique de la dynamique raffinée des T-AN, nous avons introduit une nouvelle **méthode d'inférence** (ou d'*apprentissage*) qui apprend des RRB (modélisés en T-AN) à partir des données de séries temporelles.

Cependant, la modélisation ne représente que la partie initiale pour l'étude des systèmes biologiques. L'objectif principal, dès qu'un modèle satisfaisant est appris, est de réussir à l'**analyser** et à **vérifier** ses propriétés dynamiques. De plus, les modèles peuvent être alors utilisés à des fins **prédictives**. En effet, un modèle permet de réaliser des "expériences virtuelles" qui pourraient être difficiles, longues, coûteuses, voire impossibles à faire avec le système réel en laboratoire.

Dans ce but, nous avons introduit dans cette thèse des nouvelles méthodes logiques qui analysent formellement certaines propriétés dynamiques des RRB (modélisés en AN et en T-AN). En effet, nous avons pu proposer une analyse dynamique efficace pour la vérification de la propriété d'**atteignabilité**. De plus, nous avons apporté une contribution qui va dans le même sens de la compréhension de la dynamique des RRB par l'identification de ses **attracteurs** (singletons et cycliques).

Il s'est avéré que la programmation logique et plus précisément l'**Answer Set Programming** (**ASP**), est particulièrement adaptée à la résolution de problèmes combinatoires complexes de recherche (NP-complet) (Baral, 2008). Par ailleurs, l'utilisation d'ASP est considérée comme innovante dans le domaine de l'apprentissage des modèles et de l'analyse des propriétés dynamiques du modèle. Et ainsi, toutes nos nouvelles méthodes présentées dans cette thèse sont implémentées avec des nouvelles approches logiques en ASP.

Dans la suite, nous proposons à la section 7.2 un bref rappel des principaux résultats obtenus. Puis, nous ouvrons à la section 7.3 un certain nombre de perspectives dans la lignée de nos travaux.

7.2 Récapitulatif

Nous récapitulons dans cette section, les trois contributions majeures de cette thèse dont le principal objectif est d'étudier la dynamique des systèmes concurrents. Si notre objectif premier a été d'appliquer ces nouvelles méthodes à la biologie, il s'avère qu'elles pourraient aussi être utiles dans d'autres domaines.

(A) L'intégration des paramètres temporels dans les RRB modélisés en AN : l'introduction des T-AN.

Dans plusieurs systèmes biologiques (e.g., le système de l'horloge circadienne), il est parfois essentiel d'avoir une information sur le temps qui sépare ses deux événements. Autrement dit, avoir une information sur la période du temps écoulée entre deux états distincts observables du RRB qui modélise ce système. Nous avons alors présenté dans le chapitre 3, une nouvelle approche de modélisation qui introduit une composante temporelle, appelée **le délai** qui est spécifique à chaque interaction dans un RRB. En effet, ce délai indique la durée entre l'instant auquel l'interaction commence son activité et l'instant auquel l'effet (activation ou inhibition) est complètement produit sur le composant qu'elle cible.

Nous modélisons dans cette thèse les RRB avec le formalisme des AN. Ainsi, l'intégration des délais est réalisée pour chaque transition locale des automates du réseau. Nous définissons un nouveau formalisme appelé T-AN. La particularité de notre modélisation via les T-AN est le raffinement qu'elle apporte à la dynamique des AN.

Dans ce but de **raffinement de la dynamique**, nous avons développé l'impact de cet enrichissement par la composante temporelle sur la dynamique du modèle. Il s'est avéré, comme il a été montré dans ce manuscrit, que pendant la période de temps qu'une transition locale d'un automate est activée, des *conflits* avec d'autres transitions locaux du T-AN peuvent se créer. Un conflit est dû principalement aux ressources partagées entre les transitions locaux. En effet, pour qu'une transition locale puisse se produire, elle nécessite d'autres éléments du système mais si ces éléments, ou ces ressources, sont aussi consommés par une autre transition locale simultanément, ceci peut induire des conflits entre elles. Ainsi, dans le but de résoudre ces conflits tout en raffinant la dynamique des AN, nous avons présenté dans le chapitre 3, une nouvelle sémantique de la dynamique des T-AN. Et grâce à cette nouvelle sémantique de la dynamique des T-AN, des nouvelles évolutions de la dynamique peuvent être observées et d'autres peuvent être supprimées. Par conséquent, nous capturons un comportement plus réaliste et donc plus proche (i.e., moins généraliste et donc plus raffiné) du comportement du système biologique modélisé.

(B) L'inférence des T-AN à partir des données de séries temporelles.

Les données de séries temporelles illustrent l'évolution de la dynamique d'un système biologique au cours du temps. Plus précisément, elles montrent l'évolution de l'expression des composants du système au cours du temps. Nous avons proposé alors, dans le chapitre 4, une nouvelle méthode qui **automatise l'inférence des RRB** à partir des données de séries temporelles du système biologique à modéliser. Nous l'appelons : `MoT-AN` (Modélisation des T-AN) car les modèles des RRB qu'elle apprend sont présentés avec le formalisme des T-AN.

Comme il est montré dans le chapitre 4, l'originalité de cette technique réside principalement dans l'enrichissement des graphes des interactions à partir des données de séries temporelles par : (i) l'intégration directe des délais quantitatifs dans l'approche d'inférence ; (ii) l'identification des niveaux discrets qualitatifs des composants qui participent à chaque interaction ; et (iii) l'identification du signe des interactions (activation ou inhibition).

En outre, nous avons présenté dans le chapitre 4, une autre approche qui s'inspire de `MoT-AN` (appelée `RevT-AN`) qui a comme objectif de **réviser** les T-AN existants tant qu'ils ne sont pas cohérents avec les données de séries temporelles nouvellement fournies

(c'est-à-dire différentes de celles à partir desquelles les T-AN ont été appris). Ainsi, s'il y a des transitions locales temporisées manquantes et qui doivent exister dans le T-AN pour modéliser correctement le système, `RevT-AN` les ajoutera dans le nouveau T-AN révisé.

Cette approche de révision fait aussi partie du processus de **validation** des modèles après leur apprentissage. En effet, parfois l'apprentissage est réalisé à partir de plusieurs observations du système : avec et sans perturbations (e.g., suppression ou ajout d'un composant).

De plus, nous avons proposé un raffinement des modèles T-AN appris par l'application d'un ensemble de **filtres** qui permettent, entre autres, de réviser les T-AN appris. En effet, les filtres introduits dans cette thèse améliorent le résultat de la méthode d'inférence des T-AN (i.e., de `MoT-AN`). Parmi ces filtres, il y en a un qui permet d'enlever les incohérences, si elles existent, entre la dynamique du T-AN appris et la dynamique du système modélisé. Un autre filtre qui compare tous les T-AN appris entre eux et élimine ceux qui sont moins probablement corrects. Nous avons aussi introduit un autre filtre qui essaie de faire face aux informations incorrectes qui sont apprises à partir des données bruitées.

(C) L'analyse formelle des propriétés dynamiques dans les RRB

Nous avons proposé dans le chapitre 5 des méthodes qui réalisent une analyse formelle et exhaustive des propriétés dynamiques des RRB. En effet, ces méthodes assurent une meilleure compréhension de la dynamique du système biologique modélisé par le RRB et aussi elles permettent de déchiffrer leurs comportements complexes.

La vérification de la propriété d'atteignabilité

L'étude la propriété d'atteignabilité consiste à répondre à la question suivante : "existe-t-il un chemin dans le modèle qui permet d'atteindre un certain objectif à partir d'un état global initialement connu ?". Nous avons vérifié cette propriété dynamique dans les RRB modélisés en AN et en T-AN. Et donc, l'objectif fixé pour la propriété d'atteignabilité est un ensemble d'états locaux des différents automates du réseau.

Pour calculer les chemins dans les RRB modélisés en T-AN, nous nous sommes basés sur la sémantique que nous avons proposée dans le chapitre 3. Et pour les AN, l'étude que nous avons présentée permet de calculer leurs chemins selon la sémantique *synchrone* ou *asynchrone* non-déterministe.

Nous avons alors pu mettre en œuvre une nouvelle approche logique (implémentée en ASP), dont l'originalité réside principalement dans l'énumération exhaustive de tous les chemins possibles (d'une certaine longueur) qui permettent d'atteindre l'objectif de la propriété d'atteignabilité. De plus, en nous appuyant sur les fonctionnalités des options d'ASP, nous avons adapté cette méthode pour réussir à identifier seulement les plus courts chemins.

L'identification des attracteurs cycliques et singletons

Un *attracteur* est un domaine piège minimal, c'est-à-dire une partie du graphe d'états dont la dynamique ne peut pas s'échapper. Ces structures permettent une compréhension approfondie de la dynamique des RRB car ils représentent une caractéristique importante du système modélisé. En effet, de point du vue pratique et applicatif, certain attracteurs

peuvent par exemple être révélateurs du passage dans un mode pathologique et de façon définitive.

Nous avons montré dans le chapitre 5 que les attracteurs peuvent être des singletons appelés aussi des **points fixes** ou un ensemble d'états présentant une dynamique cyclique entre eux. En outre, nous avons présenté dans ce chapitre 5, une étude théorique approfondie concernant ces *attracteurs cycliques*. Ensuite, nous avons introduit dans le chapitre 6 deux nouvelles méthodes logiques différentes (développées en ASP) pour identifier les deux types d'attracteurs (singletons et cycliques).

Simulation : validation et prédiction de la dynamique du modèle

Nous avons mis en œuvre, dans le chapitre 6, des méthodes de simulation qui respectent les hypothèses que nous avons adoptées par rapport à la sémantique de la dynamique des AN (synchrone et asynchrone) d'une part, et celle des T-AN d'autre part (introduite dans le chapitre 3). Le but de ces simulateurs est de faire évoluer le modèle afin de pouvoir identifier sa dynamique au cours du temps (c'est-à-dire identifier les trajectoires qui pourront être suivies par le modèle). Et une fois que ces trajectoires sont trouvées, elles sont utilisées, d'une part, pour valider le modèle appris (en comparant ce qui est trouvé à ce qui est attendu), et d'autre part, pour la vérification des propriétés dynamiques du modèle.

Le fait de pouvoir identifier les différentes évolutions possibles de la dynamique du modèle offre la possibilité de prédire de nouveaux comportements du système modélisé. Autrement dit, il s'agit de trouver un comportement qui n'est pas observé dans les résultats expérimentaux (c'est-à-dire non utilisé pour apprendre le modèle).

Application

Enfin, le chapitre 6 nous a permis d'illustrer l'étendue des différents résultats obtenus dans cette thèse par leurs applications sur des données et des modèles des systèmes biologiques réels.

Parmi les systèmes traités, nous citons principalement les données de séries temporelles du E.coli (DREAM4) (Prill et al., 2011) et du cancer du sein (DREAM8) (Hill et al., 2016) utilisées pour la validation de la méthode *MoT-AN* de l'inférence des T-AN. En outre, à la fin du chapitre 4, nous avons interprété les données de séries temporelles du système de l'horloge circadienne du foie comme une application directe de la méthode *MoT-AN*.

Ensuite, nous avons traité d'autres exemples d'application comme le contrôle immunitaire du bactériophage lambda (Thieffry & Thomas, 1995), le cycle cellulaire des mammifères (Fauré et al., 2006), et la différenciation cellulaire (Abou-Jaoudé et al., 2014)... Sur les modèles de ces systèmes biologiques, nous avons appliqué nos nouvelles approches sur l'analyse formelle des propriétés dynamiques : la vérification de l'atteignabilité et l'identification des attracteurs. Nous avons ainsi montré que nous étions capables d'obtenir des résultats remarquables, y compris pour des systèmes de très grandes tailles (e.g., le T-helper a plus de 100 composants).

7.3 Perspectives de travail

Les travaux présentés au cours de cette thèse ouvrent plusieurs pistes de travail permettant d'étendre et d'exploiter les résultats obtenus.

7.3.1 Extensions de l'analyse dynamique

Nous avons présenté dans le chapitre 5, des analyses dynamiques exhaustives de deux propriétés principales : la vérification de l'atteignabilité et l'identification des attracteurs dynamiques.

En ce qui concerne la méthode qui vérifie la propriété d'atteignabilité, elle ne permet actuellement de vérifier que des propriétés qui s'expriment en logique CTL sous la forme **EF**(P) (c'est-à-dire qui marque l'existence d'un chemin conduisant à un état où la propriété P est vraie). Il pourrait être intéressant de l'étendre avec de **nouveaux types de propriétés** comme **AF**(P), assurant par exemple qu'un état où P est vraie, est toujours atteignable quel que soit le chemin. Ou encore des propriétés qui permettraient d'observer des bifurcations de la dynamique ; par exemple, la propriété (P **EU** Q) dit qu'il existe un chemin tel que P est vraie jusqu'à ce que Q soit vraie (ici la bifurcation est observée de la propriété P vers Q).

L'analyse d'atteignabilité que nous avons développée est applicable sur les RRB modélisés en AN ou en T-AN. Les différents chemins trouvés pour les T-AN présentent surement des différences de durée pour atteindre l'objectif de la propriété d'atteignabilité. Ainsi, notre but est d'étudier davantage tous ces chemins trouvés afin d'identifier la **période de temps minimale ou maximale** pour atteindre l'objectif. En effet, le plus court chemin (c'est-à-dire le chemin qui active un nombre minimal de transitions) n'est pas nécessairement le chemin qui a la plus courte durée de temps pour s'exécuter. Par exemple, une seule transition locale temporisée dont le délai est égal à 10 prendrait plus de temps à s'exécuter que 3 transitions dont le délai de chacune est égal à 2 (2*3=6<10).

De plus, cette information supplémentaire sur les délais pour les transitions locales temporisées des T-AN pourrait nous permettre d'avoir une autre information qui porte sur les probabilités de l'activation des transitions dans un état global donné (Fippo-Fitime, 2016). Ainsi, ceci présente une nouvelle perspective dont le but est de calculer **la probabilité d'atteindre un objectif** par rapport à un autre dans les T-AN.

Nous avons réussi à proposer dans cette thèse, de nouvelles approches logiques qui identifient les attracteurs (cycliques et singletons) dans les AN. Par contre, pour les T-AN, nous avons seulement proposé une méthode pour identifier les attracteurs singletons (c'est-à-dire les points fixes). L'un de nos objectifs est donc d'étendre ce travail de recherche des **attracteurs cycliques dans les T-AN.**

En outre, une fois que les attracteurs sont identifiés dans un AN (ou un T-AN) il serait intéressant de pouvoir aussi identifier les **états transitoires** à partir desquels l'activation d'une transition ou d'une autre amène inévitablement le modèle vers des domaines pièges distincts et par la suite vers des attracteurs différents. En effet, nous pouvons considérer que ces états transitoires sont des éléments clés dans la dynamique du modèle car ils font le choix de le faire évoluer vers des états à partir desquels il n'est plus possible de revenir en arrière.

7.3.2 Raffinement de l'apprentissage et de la révision des RRB

La construction de modèles adaptés aux propriétés temporelles est un problème intéressant et c'est un sujet difficile dans le domaine de la biologie des systèmes. Nous avons ainsi mis en valeur dans le chapitre 4, une nouvelle approche logique qui apprend des RRB,

présentés avec le formalisme des T-AN, à partir des données de séries temporelles décrivant la dynamique du système à modéliser. Ensuite, nous proposons de raffiner les modèles appris par un ensemble de filtres qui se basent principalement sur les données de séries temporelles.

Habituellement, les biologistes connaissent certaines **propriétés dynamiques du système à modéliser** : des attracteurs ou des cycles ou même des chemins de signalisation spécifiques... Ainsi, une autre approche applicable de raffinement des modèles appris consiste à vérifier si le modèle satisfait ces propriétés dynamiques initialement connues. En effet, l'une de nos perspectives est d'être capable d'**utiliser ces propriétés dynamiques lors de l'apprentissage des RRB**. Quand notre approche d'inférence pourrait profiter des connaissances préexistantes sur le système à modéliser, elle pourrait être encore plus efficace.

7.3.3 La résilience du modèle de RRB

Dans le but de s'assurer qu'un modèle représentant un système biologique est assez robuste contre les perturbations, une question concernant la propriété de sa **résilience** est importante. En effet, cette question concerne sa capacité et sa propension à revenir vers un état acceptable après un certain changement brutal dans le modèle ; par exemple, la suppression ou l'ajout d'un nouveau composant ou encore d'une nouvelle transition dans le modèle. Cette **résilience** peut être aussi vue comme un moyen pour valider la cohérence du modèle avec le système biologique réel qu'il représente par la vérification de sa capacité à supporter ces perturbations.

Ainsi, les principales nouvelles questions qui se posent sont les suivantes : le modèle court-il le risque d'être forcé à atteindre un état non acceptable ? Est-il capable de revenir à tout moment dans un état acceptable ? Et à quel prix ? Répondre à ces questions pourra nécessiter d'élargir le cadre des propriétés vérifiables de manière efficace. Il serait nécessaire d'observer les conséquences sur la dynamique des **perturbations ponctuelles aléatoires** dans un modèle. Alors, il parait utile d'avoir recours à des approches différentes telles que le calcul des **cut-sets** (voir par exemple (Paulevé et al., 2013)) afin d'identifier des parties sensibles des modèles, en révélant les parties du modèle les plus sensibles aux perturbations.

7.3.4 Enrichissement de la modélisation des RRB

Il est possible d'envisager d'autres extensions de la sémantique de la dynamique des T-AN afin de raffiner la dynamique des AN après l'intégration des délais dans leurs transitions locales. Par exemple, il serait possible de considérer qu'au cours de l'activation d'une transition locale temporisée, les niveaux locaux des automates ne restent pas constants comme c'est le cas de la sémantique des T-AN que nous avons présentée au chapitre 3. Autrement dit, durant la période d'activation d'une transition locale temporisée, l'automate qu'elle cible change son niveau après chaque unité de temps en fonction du délai de cette transition (voir l'exemple ci-dessous qui illustre l'évolution des niveaux des automates sous la forme des **rationnels**).

Exemple. Soit $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$ un T-AN, tel que $\Sigma = \{a, b, c\}$ et $\tau_1, \tau_2 \in \mathcal{T}$ avec $\tau_1 = a_1 \xrightarrow[5]{b_1} a_0$ et $\tau_2 = c_0 \xrightarrow[3]{\emptyset} c_1$.

Supposant qu'à $t = 0$, $\mathbf{état}(\mathcal{AN}, 0) = \langle a_1, b_1, c_0 \rangle$ donc τ_1 et τ_2 sont franchissables à $t = 0$. Rappelons qu'une transition locale temporisée $\tau = a_i \xrightarrow[\delta]{\ell} a_j$ est franchissable (définition 3.11 en page 71) à un instant t, si et seulement si, elle est jouable dans un état global ζ (c'est-à-dire $a_i = \zeta[a]$ et $\forall b_k \in \ell$, $b_k = \zeta[b]$ tel que $\zeta[a]$ est l'état local de a dans l'état global ζ) et qu'elle n'est pas en conflit avec aucune autre transition en cours.

Nous donnons ci-dessous, un exemple d'une trajectoire de ce T-AN pendant la période d'activation de τ_1 (i.e., dans $[0, \mathsf{delai}(\tau_1)] = [0, 5]$) et de τ_2 (i.e., dans $[0, \mathsf{delai}(\tau_2)] = [0, 3]$). Nous y montrons alors l'évolution des niveaux discrets des automates qui sont plutôt des rationnels.

t	0	1	2	3	4	5

$$\langle a_1, b_1, c_0 \rangle \xrightarrow[\tau_1 + \tau_2]{} \langle \boldsymbol{a_{\frac{4}{5}}}, b_1, \boldsymbol{c_{\frac{1}{3}}} \rangle \xrightarrow[\tau_1 + \tau_2]{} \langle \boldsymbol{a_{\frac{3}{5}}}, b_1, \boldsymbol{c_{\frac{2}{3}}} \rangle \xrightarrow[\tau_1 + \tau_2]{} \langle \boldsymbol{a_{\frac{2}{5}}}, b_1, \boldsymbol{c_1} \rangle \xrightarrow[\tau_1]{} \langle \boldsymbol{a_{\frac{1}{5}}}, b_1, c_1 \rangle \xrightarrow[\tau_1]{} \langle \boldsymbol{a_0}, b_1, c_1 \rangle$$

On peut remarquer qu'après chaque unité de temps, le niveau de l'automate a diminue de $\frac{1}{5}$ car $\mathsf{delai}(\tau_1) = 5$ et celui de l'automate c, augmente de $\frac{1}{3}$ car $\mathsf{delai}(\tau_2) = 3$.

Ainsi, il serait intéressant de considérer qu'au cours de l'activation des transitions locales temporisées, les niveaux locaux des automates modifiés soient plutôt des rationnels ($x \in \mathbb{Q}_+$) et ne restent pas constants (des entiers). Par exemple, si une transition locale temporisée $\tau = a_i \xrightarrow[\delta]{\ell} a_j$ est activée à un instant t, alors a ne reste plus au niveau i pendant la période d'activation de τ, c'est-à-dire dans l'intervalle de temps $[t, t + \delta]$ mais il change en fonction du sens de la variation (de i vers j) et aussi en fonction du délai δ. Et donc, à chaque instant $t' \in [t, t + \delta]$, l'automate a a un niveau qui est égal à $\zeta_{t'}[a] = a_x$ tel que $x \in \mathbb{Q}_+$ (rappelons que nous notons par ζ_t l'état global du réseau à l'instant t).

Pour faciliter l'introduction de cette perspective, nous avons supposé qu'après chaque unité de temps, si un composant dont le niveau est en train d'être changé par une transition τ, alors ce dernier diminue (si c'est une inhibition) ou augmente (si c'est une activation) de $\frac{1}{\mathsf{delai}(\tau)}$. Ainsi, pour $\zeta_t[a] = a_n$, $\forall t' \in [t, t + \delta]$ on a $\zeta_{t'}[a] = a_x$ tel que $x = n + \frac{1}{\mathsf{delai}(\tau)}$ si $i < j$ ou $x = n - \frac{1}{\mathsf{delai}(\tau)}$ si $i > j$.

Nous pensons qu'il est intéressant de pouvoir calculer la dynamique des T-AN avec une telle sémantique qui retourne de telles trajectoires avec des rationnels. Cela montre plus concrètement l'évolution des niveaux discrets de tous les automates. En effet, parmi les avantages d'une telle trajectoire, figure le fait en l'observant, il est possible de distinguer les automates qui sont en train d'être inhibés, et ceux qui sont en train d'être activés et aussi ceux qui ne subissent aucune modification (i.e., stagnants). En occurrence, pour l'exemple ci-dessus, on peut observer a qui est en train d'être inhibé c en train d'être activé alors que b stagnant.

Ainsi, on peut dire qu'une telle sémantique – dans laquelle les niveaux discrets des automates changent au cours de leur activation (ou inhibition) sous la forme de rationnels – pourrait présenter une dynamique qui s'approche beaucoup de la dynamique présentant une évolution continue des niveaux d'expression des composants. Réussir à la mettre en œuvre présente un intéressant prolongement des travaux entamés dans cette thèse. Nous

discutons alors, dans la suite, de quelques nouvelles pistes de recherche envisageables, dans la même lignée.

Rappelons que les niveaux discrets (qui sont des entiers) d'un automate sont trouvés par l'abstraction de l'expression (i.e., concentration) du composant qu'ils présentent dans le système. Ce processus est appelé la **discrétisation** (voir section 4.2.1.1 en page 86). La discrétisation et basée principalement sur le choix des seuils tels que chaque seuil sépare entre deux niveaux discrets d'un composant ($\{0,1,\dots\}$). Quand on dit dans un T-AN que a est au niveau 1, cela signifie que a un niveau d'expression qui est supérieur ou égal au seuil choisi pour séparer entre ses deux niveaux 0 et 1. Ainsi, la transition suivante $\tau = a_0 \xrightarrow[5]{b_1} a_1$, exprime le fait que quand le niveau d'expression de a est au-dessous de son seuil, il va commencer à augmenter pour atteindre ce seuil grâce à b qui est effectivement au-dessus de son propre seuil.

En revanche, puisque la discrétisation n'est pas toujours parfaite, alors il peut y avoir des erreurs dans le choix des seuils. Par exemple, dans la transition $\tau = a_0 \xrightarrow[5]{b_1} a_1$, peut-être que b ne peut agir sur a que quand il dépasse un peu ce seuil ou peut être même avant de l'atteindre et non pas au moment exact où il l'atteint. Il serait donc intéressant de prendre en compte **une marge d'erreur** ($e \in]0,1[$) quand nous étudions la dynamique des modèles discrets (dans notre cas les T-AN). Cette marge d'erreur permettrait alors d'exprimer ainsi la transition τ : si b est au niveau $1 \pm e$ et si a est au niveau $0 \pm e$, alors a évolue pour changer son niveau vers $1 \pm e$.

Dans une sémantique de la dynamique où nous n'utilisons que des entiers pour exprimer les niveaux des composants, il n'est pas possible de considérer cette marge d'errer. En effet, il n'est pas possible de vérifier si les niveaux des composants sont égaux à $1 \pm e$ ou à $0 \pm e$. En revanche, si nous considérons une sémantique dont l'évolution des niveaux des automates change sous la forme de rationnels ce sera possible. Nous illustrons ceci dans l'exemple suivant.

Exemple. Soit $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$ un T-AN, tel que $\Sigma = \{a, b, c\}$ et $\tau_1, \tau_2 \in \mathcal{T}$ avec $\tau_1 = a_1 \xrightarrow[5]{b_1} a_0$, $\tau_2 = c_0 \xrightarrow[3]{a_1} c_1$. Soit $e = 0.2$ une marge d'erreur.

t	0	1	2	3	4	5
	$\langle a_0, b_1, c_0\rangle \xrightarrow[\tau_1]{} \langle \boldsymbol{a}_{\frac{1}{5}}, b_1, c_0\rangle$	$\xrightarrow[\tau_1]{} \langle \boldsymbol{a}_{\frac{2}{5}}, b_1, c_0\rangle$	$\xrightarrow[\tau_1]{} \langle \boldsymbol{a}_{\frac{3}{5}}, b_1, c_0\rangle$	$\xrightarrow[\tau_1]{} \langle \boldsymbol{a}_{\frac{4}{5}}, b_1, c_0\rangle$	$\xrightarrow[\tau_1 ?]{} \langle \boldsymbol{a}_1, b_1, c_0\rangle$	
					$\tau_1 + \tau_2 ?$	$\langle \boldsymbol{a}_1, b_1, \boldsymbol{c}_{\frac{1}{3}}\rangle$

Nous montrons ci-dessus deux simulations d'évolution de la dynamique (i.e., deux trajectoires) du T-AN considéré pendant 5 unités de temps. La marge d'erreur ne peut être prise en compte qu'à $t = 4$, En effet, a atteint le niveau $\frac{4}{5} = 0.8$ et qui est égal à $1 - e = 1 - 0.2$. Et donc la transition $\tau_2 = c_0 \xrightarrow[\delta]{\{a_1\}} c_1$ pourrait être considérée comme jouable à $t = 4$ car le niveau de a est égal à $1 \pm e$ (i.e., $1 - e$). Ainsi, à la trajectoire de la deuxième ligne, c commence à être activée mais ce n'est pas le cas pour la trajectoire de la première ligne. Rappelons que pour la sémantique des T-AN présentée dans le chapitre 3,

nous considérons que les niveaux sont seulement de type des entiers, et qu'il faut attendre que a atteint effectivement le niveau 1 pour que τ_2 soit jouable (c'est le cas de la première trajectoire).

Finalement, laquelle des trajectoires serait plus proche de la dynamique réelle du système biologique modélisé ? Il serait intéressant d'approfondir la recherche dans cette perspective qui a comme but de raffiner les informations apprises après la discrétisation.

Par conséquent, et dans cette même volée de perspectives, il serait peut-être intéressant d'aller vers la définition de **transitions locales temporisées applicables par intervalles**. Autrement dit, il s'agit d'adapter la définition d'une transition locale temporisée jouable tout en considérant la marge d'erreur pour les niveaux locaux actifs des automates qui y participent : c'est-à-dire de $\mathsf{ori}(\tau)$ et de $\mathsf{cond}(\tau)$. En occurrence, pour l'exemple précédent, les transitions τ_1 et τ_2 seront écrites comme suit avec une marge d'erreur $e = 0.2$:
$\tau_1 = a_{0\pm e} \xrightarrow[10]{b_{1\pm e}} a_{1\pm e}$, $\tau_2 = c_{0\pm e} \xrightarrow[\delta]{\{a_{1\pm e}\}} c_{1\pm e}$

ou avec des intervalles :
$\tau_1 = a_{[0,0.2]} \xrightarrow[10]{b_{[0.8,1.2]}} a_{[0.8,1.2]}$, $\tau_2 = c_{[0,0.2]} \xrightarrow[\delta]{\{a_{[0.8,1.2]}\}} c_{[0.8,1.2]}$.

Nous notons que si un automate a n'a que deux niveaux discrets 0 et 1, alors avec des transitions locales temporisées applicables par intervalles, son état local a_1 sera exprimé ainsi : $a_{[0.8,1]}$ (c'est-à-dire l'intervalle doit être majoré par 1).

Ainsi, plusieurs questions se posent : est-ce qu'en simulant, il faudrait prendre cette marge d'erreur pour tous les composants ou non ? Est-ce-qu'il serait mieux de définir des différentes marges d'erreur pour chaque composant ? Quelle sont les hypothèses qui doivent être faites afin de pouvoir simuler un T-AN avec une telle sémantique de la dynamique où les niveaux des composants sont exprimés par des rationnels ? Quel outil serait-il le plus adapté ? Est-ce-qu'il y aurait de nouveaux conflits créés entre les transitions locales temporisées applicables par intervalles qui partagent les mêmes ressources (à part ceux qui sont présentés dans cette thèse définition 3.8 en page 68) ?

Effectivement des nouveaux conflits peuvent être crées. À titre d'exemple, nous en détaillons un ci-dessous. Il s'agit du cas où les transitions locales temporisées applicables par intervalles partageant les mêmes ressources ont des intervalles qui se chevauchent.

Exemple. Soit $\mathcal{AN} = (\Sigma, \mathcal{S}, \mathcal{T})$ un T-AN, tel que $\Sigma = \{a, b, c\}$ et $\tau_1, \tau_2, \tau_3 \in \mathcal{T}$ avec $\tau_1 = a_{[0,0.2]} \xrightarrow[12]{b_{[0.8,1]}} a_{[0.8,1]}$, $\tau_2 = c_{[0,0.2]} \xrightarrow[3]{\emptyset} c_{[0.8,1]}$ et $\tau_3 = a_{[0.8,1]} \xrightarrow[2]{c_{[0.8,1]}} a_{[0,0.2]}$.

Nous notons qu'il y a une concurrence entre τ_1 et τ_3 car les signes de leurs influences sur l'automate a sont opposés : τ_1 l'active et τ_2 l'inhibe.

Soit $\zeta_0 = \langle a_0, b_1, c_0 \rangle$ l'état initial du réseau à $t = 0$. Ci-dessous, une trajectoire du réseau pendant 10 unités de temps :

t	0	1	2	3	⋯	10	11

$$\langle a_0, b_1, c_0\rangle \xrightarrow[\tau_1+\tau_2]{} \langle \boldsymbol{a}_{\frac{1}{12}}, b_1, \boldsymbol{c}_{\frac{1}{3}}\rangle \xrightarrow[\tau_1+\tau_2]{} \langle \boldsymbol{a}_{\frac{2}{12}}, b_1, \boldsymbol{c}_{\frac{2}{3}}\rangle \xrightarrow[\tau_1+\tau_2]{} \langle \boldsymbol{a}_{\frac{3}{12}}, b_1, \boldsymbol{c_1}\rangle \xrightarrow[\tau_1]{} \cdots \xrightarrow[\tau_1]{} \langle \boldsymbol{a}_{\frac{10}{12}}, b_1, c_1\rangle \xrightarrow[?]{} ?$$

Le conflit entre τ_1 et τ_3 n'est apparu qu'à $t = 10$, car à cet instant, τ_1 est en cours et τ_3 est devenu jouable. En effet, le niveau actif de a est égal à $\frac{10}{12} = 0.833 \in [0.8, 1[$ et on a

c qui est au niveau 1, ainsi, $\tau_3 = a_{]0.8,1]} \overset{c_{]0.8,1]}}{\underset{2}{\rightarrow}} a_{[0,0.2[}$ est jouable dans cet état. En revanche, puisqu'il y a τ_1 qui est en cours, la question sera de savoir si τ_3 est franchissable.

Nous rappelons que dans de telles situations, la sémantique des T-AN présentée dans cette thèse cède la priorité à la transition locale temporisée qui est en cours. En revanche, l'enrichissement de cette sémantique serait important en définissant par exemple des **classes prioritaires** pour les transitions locales temporisées. Ainsi, celles qui ont une priorité plus élevée pourraient interrompre celles qui sont en cours dont la priorité est inférieure.

Il existe aussi une autre possibilité pour résoudre ce conflit par la **comparaison entre les durées** mises par toutes les transitions qui sont en conflit pour terminer leur activité. Et ainsi, laisser celle qui est plus rapide (c'est-à-dire qui se termine la première) à s'activer. En revanche, dans une telle sémantique de la dynamique, qui est assez complexe à étudier, serait-il facile d'identifier la valeur exacte de la durée du temps nécessaire pour qu'une transition locale temporisée applicable par intervalles se termine ? La réponse est probablement non ; car cette durée que mettra, par exemple, une transition τ, ne dépend pas seulement de son délai ($\mathsf{delai}(\tau)$) mais aussi de tous les automates qui y participent (i.e., de $\mathsf{ori}(\tau)$ et de $\mathsf{cond}(\tau)$).

En effet, cette durée dépend, en premier lieu, du niveau discret rationnel de l'automate de son origine ($\mathsf{ori}(\tau)$) au moment de l'activation de τ. Ceci est du au fait que la transition pourrait être activée à partir des différents niveaux discrets rationnels de l'origine (e.g., $\forall x \in [0, 0.2[$). Par exemple, la durée qu'une transition mettra pour faire évoluer un automate a du niveau $\frac{10}{12}$ vers le niveau 1 ne sera pas égale à la durée qu'elle mettra pour le faire évoluer de $\frac{3}{10}$ vers 1.

Ainsi, quand une transition locale temporisée applicable par intervalles ayant un délai δ, est activée à un instant t, son résultat ne sera pas nécessairement observé à l'instant $t + \delta$. En revanche, c'est le cas pour la sémantique des T-AN proposée dans le chapitre 3 où nous considérons que les transitions ne peuvent pas commencer leur activité avant d'atteindre l'unique niveau discret qui est de type entier. Et donc, la durée mise par la transition locale temporisée pour se terminer est exactement égale à son délai δ (et elle se termine toujours à l'instant $t + \delta$ quand elle commence à l'instant t).

La durée mise par τ, une transition locale temporisée applicable par intervalles, pour se terminer selon une telle sémantique de la dynamique où les niveaux des composants sont des rationnels, dépendra aussi des niveaux de tous les automates qui font partie de sa condition (i.e., de $\mathsf{cond}(\tau)$). En effet, on pourrait considérer que pour des différents niveaux de ces automates, la transition aura des durées différentes. Par exemple, pour la transition $\tau_1 = a_{[0,0.2[} \overset{\{b_{]0.8,1]}\}}{\underset{12}{\rightarrow}} a_{]0.8,1]}$, logiquement, la période de temps que mettra τ_1 pour activer a afin de le faire évoluer d'un niveau inclus dans [0,0.2[vers un niveau inclus dans]0.8,1] ne sera pas la même si le niveau de b est égal à 0.82 ou s'il est égal à 1. Car dans plusieurs cas, si l'activateur (ou l'inhibiteur) est plus abondant et s'il ne participe pas à d'autres interactions au même temps, son influence sur le composant qu'il modifie sera plus rapide. Et c'est le cas de b quand il est au niveau 1, il est considéré plus abondant donc, probablement, il activera a plus rapidement.

Ainsi, puisque l'une de nos perspectives consiste à considérer cette sémantique de la dynamique où les niveaux des automates sont présentés avec des rationnels, il faudrait trouver une fonction qui calcule la durée nécessaire pour qu'une transition locale temporisée

applicable par intervalles ait lieu. Cette fonction doit prendre en compte (a) le délai de la transition ($\mathsf{delai}(\tau)$) ; (b) le niveau discret rationnel de l'automate de son origine ($\mathsf{ori}(\tau)$) ; (c) et aussi des niveaux discrets rationnels de tous les automates qui y participent ($\mathsf{cond}(\tau)$).

Notre objectif est de trouver **une sémantique de la dynamique des T-AN qui serait plus raffinée** et donc plus proche de la dynamique du système modélisé. En effet, cette sémantique avec des rationnels pourrait permettre au modélisateur de proposer une évolution de la dynamique du modèle qui s'approche de l'évolution continue. Il serait donc intéressant d'approfondir les recherches dans cette direction et de raffiner la sémantique par une **utilisation combinée des entiers et des rationnels**.

Bibliographie

Abou-Jaoudé, W., Monteiro, P. T., Naldi, A., Grandclaudon, M., Soumelis, V., Chaouiya, C. & Thieffry, D. (2014), 'Model checking to assess t-helper cell plasticity', *Frontiers in bioengineering and biotechnology* **2**.

Ahmad, J., Bernot, G., Comet, J.-P., Lime, D. & Roux, O. (2006), 'Hybrid modelling and dynamical analysis of gene regulatory networks with delays', *ComPlexUs* **3**(4), 231–251.

Ahmad, J., Roux, O., Bernot, G., Comet, J.-P. & Richard, A. (2008), 'Analysing formal models of genetic regulatory networks with delays', *International Journal of Bioinformatics Research and Applications (IJBRA)* **4**(2).

Akutsu, T., Kosub, S., Melkman, A. A. & Tamura, T. (2012), 'Finding a periodic attractor of a boolean network', *Computational Biology and Bioinformatics, IEEE/ACM Transactions on* **9**(5), 1410–1421.

Akutsu, T., Tamura, T. & Horimoto, K. (2009), Completing networks using observed data, *in* 'Algorithmic Learning Theory', Springer, pp. 126–140.

Albert, R. & Othmer, H. G. (2003), 'The topology of the regulatory interactions predicts the expression pattern of the segment polarity genes in drosophila melanogaster', *Journal of theoretical biology* **223**(1), 1–18.

Alur, R., Belta, C., Kumar, V., Mintz, M., Pappas, G. J., Rubin, H. & Schug, J. (2002), 'Modeling and analyzing biomolecular networks', *Computing in Science & Engineering* **4**(1), 20–31.

Alur, R., Courcoubetis, C. & Dill, D. (1990), Model-checking for real-time systems, *in* 'Logic in Computer Science, 1990. LICS 90, Proceedings., Fifth Annual IEEE Symposium on e', pp. 414–425.

Baier, C., Bertrand, N., Bouyer, P., Brihaye, T. & Größer, M. (2007), Probabilistic and topological semantics for timed automata, *in* 'International Conference on Foundations of Software Technology and Theoretical Computer Science', Springer, pp. 179–191.

Baier, C., Bertrand, N., Bouyer, P., Brihaye, T. & Grosser, M. (2008), Almost-sure model checking of infinite paths in one-clock timed automata, *in* 'Logic in Computer Science, 2008. LICS'08. 23rd Annual IEEE Symposium on', IEEE, pp. 217–226.

Baral, C. (2003), *Knowledge representation, reasoning and declarative problem solving*, Cambridge university press.

Baral, C. (2008), Using answer set programming for knowledge representation and reasoning : Future directions, *in* 'ICLP', pp. 69–70.

Batt, G., Salah, R. B. & Maler, O. (2007), On timed models of gene networks, *in* 'International Conference on Formal Modeling and Analysis of Timed Systems', Springer, pp. 38–52.

Behaegel, J., Comet, J.-P., Bernot, G., Cornillon, E. & Delaunay, F. (2016), 'A hybrid model of cell cycle in mammals', *Journal of bioinformatics and computational biology* **14**(01), 1640001.

Ben Abdallah, E., Folschette, M., Roux, O. & Magnin, M. (2015), Exhaustive analysis of dynamical properties of biological regulatory networks with answer set programming, *in* 'IEEE International Conference on Bioinformatics and Biomedicine (BIBM)', IEEE, pp. 281–285.

Ben Abdallah, E., Folschette, M., Roux, O. & Magnin, M. (2017), 'ASP-based method for the enumeration of attractors in non-deterministic synchronous and asynchronous multi-valued networks', *Algorithms for Molecular Biology* **11**(1).

Ben Abdallah, E., Ribeiro, T., Magnin, M., Roux, O. & Inoue, K. (2016), Inference of delayed biological regulatory networks from time series data, *in* 'International Conference on Computational Methods in Systems Biology', Springer, pp. 30–48.

Ben Abdallah, E., Ribeiro, T., Magnin, M., Roux, O. & Inoue, K. (2017), 'Modeling delayed dynamics in biological regulatory networks from time series data', *Algorithms* **10**(1), 8.

Bernot, G., Comet, J.-P. & Khalis, Z. (2008), Gene regulatory networks with multiplexes, *in* 'European Simulation and Modelling Conference Proceedings', pp. 423–432.

Bertrand, N., Bouyer, P., Brihaye, T. & Markey, N. (2008), Quantitative model-checking of one-clock timed automata under probabilistic semantics, *in* 'Quantitative Evaluation of Systems, 2008. QEST'08. Fifth International Conference on', IEEE, pp. 55–64.

Bridoux, F., Guillon, P., Perrot, K., Sené, S. & Theyssier, G. (2017), On the cost of simulating a parallel boolean automata network by a block-sequential one, *in* 'International Conference on Theory and Applications of Models of Computation', Springer, pp. 112–128.

Bryans, J., Bowman, H. & Derrick, J. (2003), 'Model checking stochastic automata', *ACM Transactions on Computational Logic (TOCL)* **4**(4), 452–492.

Bryant, R. E. (1986), 'Graph-based algorithms for boolean function manipulation', *Computers, IEEE Transactions on* **100**(8), 677–691.

Calzone, L., Fages, F. & Soliman, S. (2006), 'Biocham : an environment for modeling biological systems and formalizing experimental knowledge', *Bioinformatics* **22**(14), 1805–1807.

Chaouiya, C., Klaudel, H. & Pommereau, F. (2011), A modular, qualitative modeling of regulatory networks using petri nets, *in* 'Modeling in Systems Biology', Springer, pp. 253–279.

Chaouiya, C., Naldi, A. & Thieffry, D. (2012), Logical modelling of gene regulatory networks with ginsim, *in* 'Bacterial Molecular Networks', Springer, pp. 463–479.

Chatain, T., Haar, S., Jezequel, L., Paulevé, L. & Schwoon, S. (2014), Characterization of reachable attractors using petri net unfoldings, *in* 'International Conference on Computational Methods in Systems Biology', Springer, pp. 129–142.

Clarke, E. & Emerson, E. (2008), 'Design and synthesis of synchronization skeletons using branching time temporal logic', *25 Years of Model Checking* pp. 196–215.

Cokelaer, T., Bansal, M., Bare, C., Bilal, E., Bot, B., Chaibub Neto, E., Eduati, F., de la Fuente, A., Gönen, M., Hill, S., Hoff, B., Karr, J., Küffner, R., Menden, M., Meyer, P., Norel, R., Pratap, A., Prill, R., Weirauch, M., Costello, J., Stolovitzky, G. & Saez-Rodriguez, J. (2016), 'Dreamtools : a python package for scoring collaborative challenges [version 2 ; referees : 1 approved, 2 approved with reservations]', *F1000Research* **4**(1030).

Colange, M., Baarir, S., Kordon, F. & Thierry-Mieg, Y. (2013), Towards distributed software model-checking using decision diagrams, *in* 'Computer Aided Verification', Springer, pp. 830–845.

Comet, J.-P., Fromentin, J., Bernot, G. & Roux, O. (2010), A formal model for gene regulatory networks with time delays, *in* 'Computational Systems-Biology and Bioinformatics', Springer, pp. 1–13.

Courcoubetis, C. & Yannakakis, M. (1988), Verifying temporal properties of finite-state probabilistic programs, *in* '29th Annual Symposium on Foundations of Computer Science', IEEE, pp. 338–345.

Cousot, P. & Cousot, R. (1977), Abstract interpretation : A unified lattice model for static analysis of programs by construction or approximation of fixpoints, *in* 'Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of programming languages', ACM, New York, NY, USA, pp. 238–252.

Couvreur, J.-M. & Thierry-Mieg, Y. (2005), Hierarchical decision diagrams to exploit model structure, *in* 'International Conference on Formal Techniques for Networked and Distributed Systems', Springer, pp. 443–457.

Danos, V., Feret, J., Fontana, W. & Krivine, J. (2007), Scalable simulation of cellular signaling networks, *in* 'Asian Symposium on Programming Languages and Systems', Springer Berlin Heidelberg, pp. 139–157.

Danos, V., Feret, J., Fontana, W. & Krivine, J. (2008), Abstract interpretation of cellular signalling networks, *in* 'Verification, Model Checking, and Abstract Interpretation', Springer, pp. 83–97.

Davidich, M. I. & Bornholdt, S. (2008), 'Boolean network model predicts cell cycle sequence of fission yeast', *PloS one* **3**(2), e1672.

de Espanés, P. M., Osses, A. & Rapaport, I. (2016), 'Fixed-points in random boolean networks : The impact of parallelism in the barabási–albert scale-free topology case', *Biosystems* **150**, 167–176.

De Jong, H. (2002), 'Modeling and simulation of genetic regulatory systems : a literature review', *Journal of computational biology* **9**(1), 67–103.

De Jong, H., Gouzé, J.-L., Hernandez, C., Page, M., Sari, T. & Geiselmann, J. (2004), 'Qualitative simulation of genetic regulatory networks using piecewise-linear models', *Bulletin of mathematical biology* **66**(2), 301–340.

DeCarlo, R. A. (1989), *Linear systems : A state variable approach with numerical implementation*, Prentice-Hall, Inc.

Demongeot, J., Goles, E., Morvan, M., Noual, M. & Sené, S. (2010), 'Attraction basins as gauges of robustness against boundary conditions in biological complex systems', *PloS one* **5**(8), e11793.

Demongeot, J., Morvan, M. & Sené, S. (2008), Impact of fixed boundary conditions on the basins of attraction in the flower's morphogenesis of arabidopsis thaliana, *in* '22nd International Conference on Advanced Information Networking and Applications-Workshops', IEEE, pp. 782–789.

Dubrova, E. & Teslenko, M. (2011), 'A SAT-based algorithm for finding attractors in synchronous boolean networks', *IEEE/ACM transactions on computational biology and bioinformatics* **8**(5), 1393–1399.

Durzinsky, M., Marwan, W., Ostrowski, M., Schaub, T. & Wagler, A. (2011), 'Automatic network reconstruction using asp', *Theory and Practice of Logic Programming* **11**(4-5), 749–766.

Emerson, E. A. (1990), 'Temporal and modal logic.', *Handbook of Theoretical Computer Science, Volume B : Formal Models and Sematics (B)* **995**(1072), 5.

Fages, F., Soliman, S. & Chabrier-Rivier, N. (2004), 'Modelling and querying interaction networks in the biochemical abstract machine biocham', *Journal of Biological Physics and Chemistry* **4**, 64–73.

Fauré, A., Naldi, A., Chaouiya, C. & Thieffry, D. (2006), 'Dynamical analysis of a generic boolean model for the control of the mammalian cell cycle', *Bioinformatics* **22**(14), e124–e131.

Feret, J., Henzinger, T., Koeppl, H. & Petrov, T. (2012), 'Lumpability abstractions of rule-based systems', *Theoretical Computer Science* **431**, 137–164.

Feret, J., Koeppl, H. & Petrov, T. (2013), 'Stochastic fragments : A framework for the exact reduction of the stochastic semantics of rule-based models', *International Journal of Software and Informatics* .

Fippo-Fitime, L. (2016), Modélisation hybride, Analyse et Vérification quantitative des grands réseaux de régulation biologique, PhD thesis, École Centrale de Nantes.

Fippo-Fitime, L., Roux, O., Guziolowski, C. & Paulevé, L. (2016), Identification of bifurcations in biological regulatory networks using answer-set programming, *in* 'Constraint-Based Methods for Bioinformatics Workshop'.

Fippo-Fitime, L., Roux, O., Guziolowski, C. & Paulevé, L. (2017), 'Identification of bifurcation transitions in biological regulatory networks using answer-set programming', *Algorithms for Molecular Biology* **12**(1), 19.

Folschette, M. (2014), Algebraic Modeling of the Dynamics of Multi-scale Biological Regulatory Networks, PhD thesis, École Centrale de Nantes.

Folschette, M., Paulevé, L., Magnin, M. & Roux, O. (2013), 'Under-approximation of reachability in multivalued asynchronous networks', *Electronic Notes in Theoretical Computer Science* **299**, 33–51.

Folschette, M., Paulevé, L., Magnin, M. & Roux, O. (2015), 'Sufficient conditions for reachability in automata networks with priorities', *Theoretical Computer Science* **608**, 66–83.

Garg, A., Di Cara, A., Xenarios, I., Mendoza, L. & De Micheli, G. (2008), 'Synchronous versus asynchronous modeling of gene regulatory networks', *Bioinformatics* **24**(17), 1917–1925.

Garg, A., Mendoza, L., Xenarios, I. & DeMicheli, G. (2007), Modeling of multiple valued gene regulatory networks, *in* '2007 29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society', IEEE, pp. 1398–1404.

Gebert, J., Radde, N. & Weber, G.-W. (2007), 'Modeling gene regulatory networks with piecewise linear differential equations', *European Journal of Operational Research* **181**(3), 1148–1165.

Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T. & Wanko, P. (2016), 'Theory solving made easy with clingo 5', **52**.

Gebser, M., Sabuncu, O. & Schaub, T. (2010), An incremental answer set programming based system for finite modelcomputation, *in* 'Logics in Artificial Intelligence', Springer, pp. 169–181.

Gelfond, M. & Lifschitz, V. (1988), The stable model semantics for logic programming, *in* 'ICLP/SLP', pp. 1070–1080.

Giavitto, J.-L., Klaudel, H. & Pommereau, F. (2010), Qualitative modelling and analysis of regulations in multi-cellular systems using petri nets and topological collections, *in* 'MeCBIC'10, 4thWorkshop on Membrane Computing and Biologically Inspired Process Calculi', EPTCS, pp. 1–16.

Git, A., Dvinge, H., Salmon-Divon, M., Osborne, M., Kutter, C., Hadfield, J., Bertone, P. & Caldas, C. (2010), 'Systematic comparison of microarray profiling, real-time pcr, and next-generation sequencing technologies for measuring differential microrna expression', *Rna* **16**(5), 991–1006.

Goldstein, Y. A. & Bockmayr, A. (2013), A lattice-theoretic framework for metabolic pathway analysis, *in* 'Computational Methods in Systems Biology', Springer, pp. 178–191.

González, A., Chaouiya, C. & Thieffry, D. (2008), 'Logical modelling of the role of the hh pathway in the patterning of the drosophila wing disc', *Bioinformatics* **24**(16), i234–i240.

Gonzalez, A. G., Naldi, A., Sanchez, L., Thieffry, D. & Chaouiya, C. (2006), 'Ginsim : a software suite for the qualitative modelling, simulation and analysis of regulatory networks', *Biosystems* **84**(2), 91–100.

Guziolowski, C., Videla, S., Eduati, F., Thiele, S., Cokelaer, T., Siegel, A. & Saez-Rodriguez, J. (2013), 'Exhaustively characterizing feasible logic models of a signaling network using answer set programming', *Bioinformatics* .

Hale, J. K. & Lunel, S. M. V. (2013), *Introduction to functional differential equations*, Vol. 99, Springer Science & Business Media.

Hamez, A., Thierry-Mieg, Y. & Kordon, F. (2009), 'Building efficient model checkers using hierarchical set decision diagrams and automatic saturation', *Fundamenta Informaticae* **94**(3-4), 413–437.

Han, H., Shim, H., Shin, D., Shim, J. E., Ko, Y., Shin, J., Kim, H., Cho, A., Kim, E., Lee, T. et al. (2015), 'Trrust : a reference database of human transcriptional regulatory interactions', *Scientific reports* **5**.

Hansson, H. & Jonsson, B. (1994), 'A logic for reasoning about time and reliability', *Formal aspects of computing* **6**(5), 512–535.

Harvey, I. & Bossomaier, T. (1997), Time out of joint : Attractors in asynchronous random boolean networks, *in* 'Proceedings of the Fourth European Conference on Artificial Life', MIT Press, Cambridge, pp. 67–75.

Hecker, M., Lambeck, S., Toepfer, S., Van Someren, E. & Guthke, R. (2009), 'Gene regulatory network inference : data integration in dynamic models—a review', *Biosystems* **96**(1), 86–103.

Heiner, M., Gilbert, D. & Donaldson, R. (2008), Petri nets for systems and synthetic biology, *in* 'International School on Formal Methods for the Design of Computer, Communication and Software Systems', Springer Berlin Heidelberg, pp. 215–264.

Heljanko, K. & Niemelä, I. (2001), Answer set programming and bounded model checking., *in* 'The Association for the Advancement of Artificial Intelligence', Springer.

Hill, S. M., Heiser, L. M., Cokelaer, T., Unger, M., Nesser, N. K., Carlin, D. E., Zhang, Y., Sokolov, A., Paull, E. O., Wong, C. K. et al. (2016), 'Inferring causal molecular networks : empirical assessment through a community-based effort', *Nature methods* **13**(4), 310–318.

Hill, S. M., Nesser, N. K., Johnson-Camacho, K., Jeffress, M., Johnson, A., Boniface, C., Spencer, S. E., Lu, Y., Heiser, L. M., Lawrence, Y. et al. (2017), 'Context specificity in causal signaling networks revealed by phosphoprotein profiling', *Cell systems* **4**(1), 73–83.

Huang, S., Eichler, G., Bar-Yam, Y. & Ingber, D. E. (2005), 'Cell fates as high-dimensional attractor states of a complex gene regulatory network', *Physical review letters* **94**(12), 128701.

Ideker, T. & Sharan, R. (2008), 'Protein networks in disease', *Genome research* **18**(4), 644–652.

Inoue, K., Doncescu, A. & Nabeshima, H. (2013), 'Completing causal networks by meta-level abduction', *Machine learning* **91**(2), 239–277.

Irons, D. J. (2006), 'Improving the efficiency of attractor cycle identification in boolean networks', *Physica D : Nonlinear Phenomena* **217**(1), 7–21.

Joshi-Tope, G., Gillespie, M., Vastrik, I., D'Eustachio, P., Schmidt, E., de Bono, B., Jassal, B., Gopinath, G., Wu, G., Matthews, L. et al. (2005), 'Reactome : a knowledgebase of biological pathways', *Nucleic acids research* **33**(suppl 1), D428–D432.

Kanehisa, M., Sato, Y., Kawashima, M., Furumichi, M. & Tanabe, M. (2015), 'Kegg as a reference resource for gene and protein annotation', *Nucleic acids research* p. gkv1070.

Kauffman, S. A. (1969), 'Metabolic stability and epigenesis in randomly constructed genetic nets', *Journal of Theoretical Biology* **22**(3), 437–467.

Kauffman, S. A. (1993), *The origins of order : Self organization and selection in evolution*, Oxford university press.

Khalis, Z., Comet, J.-P., Richard, A. & Bernot, G. (2009), 'The smbionet method for discovering models of gene regulatory networks', *Genes, Genomes and Genomics* **3**(1), 15–22.

Klamt, S., Saez-Rodriguez, J. & Gilles, E. D. (2007), 'Structural and functional analysis of cellular networks with cellnetanalyzer', *BMC Systems Biology* **1**(1).

Klamt, S., Saez-Rodriguez, J., Lindquist, J. A., Simeoni, L. & Gilles, E. D. (2006), 'A methodology for the structural and functional analysis of signaling and regulatory networks', *BMC bioinformatics* **7**(1), 1.

Klarner, H., Bockmayr, A. & Siebert, H. (2015), 'Computing maximal and minimal trap spaces of boolean networks', *Natural Computing* **14**(4), 535–544.

Klaudel, H., Koutny, M., Pelz, E. & Pommereau, F. (2010), 'State space reduction for dynamic process creation', *Scientific Annals of Computer Science* **20**, 131–157.

Klemm, K. & Bornholdt, S. (2005), 'Stable and unstable attractors in boolean networks', *Physical Review E* **72**(5), 055101.

Koh, C., Wu, F.-X., Selvaraj, G. & Kusalik, A. J. (2009), 'Using a state-space model and location analysis to infer time-delayed regulatory networks', *EURASIP Journal on Bioinformatics and Systems Biology* **2009**(1), 1.

Li, R., Yang, M. & Chu, T. (2012), 'Synchronization of boolean networks with time delays', *Applied Mathematics and Computation* **219**(3), 917–927.

Liang, S., Fuhrman, S. & Somogyi, R. (1998), 'Reveal, a general reverse engineering algorithm for inference of genetic network architectures'.

LIP6/Move (libDDD), 'the libDDD environment'. `http://ddd.lip6.fr`.

Liu, T.-F., Sung, W.-K. & Mittal, A. (2004), Learning multi-time delay gene network using bayesian network framework, *in* 'Tools with Artificial Intelligence, 2004. ICTAI 2004. 16th IEEE International Conference on', IEEE, pp. 640–645.

Ma'ayan, A., Jenkins, S. L., Neves, S., Hasseldine, A., Grace, E., Dubin-Thaler, B., Eungdamrong, N. J., Weng, G., Ram, P. T., Rice, J. J. et al. (2005), 'Formation of regulatory patterns during signal propagation in a mammalian cellular network', *Science* **309**(5737), 1078–1083.

Maler, O. & Pnueli, A. (1995), 'Timing analysis of asynchronous circuits using timed automata', *Correct Hardware Design and Verification Methods* pp. 189–205.

Mandon, H., Haar, S. & Paulevé, L. (2012), Hybrid automata and\ epsilon-analysis on a neural oscillator, *in* E. Cinquemani & A. Donzé, eds, 'Hybrid Systems Biology : First International Workshop, HSB 2012, Newcastle Upon Tyne, UK', Lecture Notes in Computer Science, Springer International Publishing, pp. 58–72.

Marx, V. (2013), 'Biology : The big challenges of big data', *Nature* **498**(7453), 255–260.

Matsuno, H., Doi, A., Nagasaki, M. & Miyano, S. (2000), Hybrid petri net representation of gene regulatory network, *in* 'Pacific Symposium on Biocomputing', Vol. 5, World Scientific Press Singapore, p. 87.

Maurin, M., Magnin, M. & Roux, O. (2009), Modeling of genetic regulatory network in stochastic π-calculus, *in* 'Bioinformatics and Computational Biology', Springer Berlin Heidelberg, pp. 282–294.

Mbodj, A., Junion, G., Brun, C., Furlong, E. E. & Thieffry, D. (2013), 'Logical modelling of drosophila signalling pathways', *Molecular BioSystems* **9**(9), 2248–2258.

Mendoza, L., Thieffry, D. & Alvarez-Buylla, E. R. (1999), 'Genetic control of flower morphogenesis in arabidopsis thaliana : a logical analysis.', *Bioinformatics* **15**(7), 593–606.

Merelli, E., Rucco, M., Sloot, P. & Tesei, L. (2015), 'Topological characterization of complex systems : Using persistent entropy', *Entropy* **17**(10), 6872–6892.

Merlin, P. M. (1974), A study of the recoverability of computing systems., PhD thesis, University of California, Irvine.

Mesarović, M. D. (1968), Systems theory and biology—view of a theoretician, *in* 'Systems theory and biology', Springer, pp. 59–87.

Mushthofa, M., Torres, G., Van de Peer, Y., Marchal, K. & De Cock, M. (2014), 'ASP-G : an ASP-based method for finding attractors in genetic regulatory networks', *Bioinformatics* p. btu481.

Nakajima, N. & Akutsu, T. (2013), Network completion for time varying genetic networks, *in* 'Complex, Intelligent, and Software Intensive Systems (CISIS), 2013 Seventh International Conference on', IEEE, pp. 553–558.

Nakajima, N. & Akutsu, T. (2014*a*), 'Exact and heuristic methods for network completion for time-varying genetic networks', *BioMed research international* **2014**.

Nakajima, N. & Akutsu, T. (2014*b*), 'Network completion for static gene expression data', *Advances in bioinformatics* **2014**.

Naldi, A., Monteiro, P. T., Müssel, C., Kestler, H. A., Thieffry, D., Xenarios, I., Saez-Rodriguez, J., Helikar, T., Chaouiya, C. et al. (2015), 'Cooperative development of logical modelling standards and tools with colomoto', *Bioinformatics* p. btv013.

Niemelä, I. (1999), 'Logic programs with stable model semantics as a constraint programming paradigm', *Ann. Math. Artif. Intell.* **25**(3-4), 241–273.

Noual, M. & Sené, S. (2017), 'Synchronism versus asynchronism in monotonic boolean automata networks', *Natural Computing* pp. 1–10.

Numata, K., Imoto, S. & Miyano, S. (2008), Partial order-based bayesian network learning algorithm for estimating gene networks, *in* 'Bioinformatics and Biomedicine, 2008. BIBM'08. IEEE International Conference on', IEEE, pp. 357–360.

Paoletti, N., Yordanov, B., Hamadi, Y., Wintersteiger, C. M. & Kugler, H. (2014), Analyzing and synthesizing genomic logic functions, *in* 'Computer Aided Verification', Springer, pp. 343–357.

Paulevé, L. (2011), Modélisation, Simulation et Vérification des Grands Réseaux de Régulation Biologique, PhD thesis, École Centrale de Nantes.

Paulevé, L. (2016*a*), Goal-oriented reduction of automata networks, *in* 'International Conference on Computational Methods in Systems Biology', Springer, pp. 252–272.

Paulevé, L. (2016*b*), Pint, a static analyzer for dynamics of automata networks, *in* '14th International Conference on Computational Methods in Systems Biology (CMSB 2016)'.

Paulevé, L., Andrieux, G. & Koeppl, H. (2013), Under-approximating cut sets for reachability in large scale automata networks, *in* N. Sharygina & H. Veith, eds, 'Computer Aided Verification', Vol. 8044 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 69–84.

Paulevé, L., Chancellor, C., Folschette, M., Magnin, M. & Roux, O. (2014), *Logical Modeling of Biological Systems*, ISTE Wiley, chapter Analyzing Large Network Dynamics with Process Hitting, pp. 125–166.

Paulevé, L., Magnin, M. & Roux, O. (2011*a*), Refining dynamics of gene regulatory networks in a stochastic π-calculus framework, *in* 'Transactions on Computational Systems Biology XIII', Vol. 6575 of *Lecture Notes in Comp Sci*, Springer, pp. 171–191.

Paulevé, L., Magnin, M. & Roux, O. (2011*b*), 'Tuning temporal features within the stochastic π-calculus', *IEEE Transactions on Software Engineering* **37**(6), 858–871.

Paulevé, L., Magnin, M. & Roux, O. (2012), 'Static analysis of biological regulatory networks dynamics using abstract interpretation', *Mathematical Structures in Computer Science* **22**(04), 651–685.

Pico, A. R., Kelder, T., Van Iersel, M. P., Hanspers, K., Conklin, B. R. & Evelo, C. (2008), 'Wikipathways : pathway editing for the people', *PLoS Biol* **6**(7), e184.

Plateau, B. & Atif, K. (1991), 'Stochastic automata network of modeling parallel systems', *IEEE transactions on software engineering* **17**(10), 1093–1108.

Popova-Zeugmann, L., Heiner, M. & Koch, I. (2005), 'Time petri nets for modelling and analysis of biochemical networks', *Fundamenta Informaticae* **67**(1-3), 149–162.

Priami, C. (1995), 'Stochastic π-calculus', *The Computer Journal* **38**(7), 578–589.

Prill, R. J., Saez-Rodriguez, J., Alexopoulos, L. G., Sorger, P. K. & Stolovitzky, G. (2011), 'Crowdsourcing network inference : the dream predictive signaling network challenge', *Science signaling* **4**(189), mr7.

Qu, H., Yuan, Q., Pang, J. & Mizera, A. (2015), Improving bdd-based attractor detection for synchronous boolean networks, *in* 'Proceedings of the 7th Asia-Pacific Symposium on Internetware', ACM.

Ramchandani, C. (1973), Analysis of asynchronous concurrent systems by timed petri nets., PhD thesis, Massachusetts Institute of Technology.

Refinetti, R., Cornélissen, G. & Halberg, F. (2007), 'Procedures for numerical analysis of circadian rhythms', *Biological Rhythm Research* **38**(4), 275–325.

Remy, E., Ruet, P. & Thieffry, D. (2008), 'Graphic requirements for multistability and attractive cycles in a boolean dynamical framework', *Advances in Applied Mathematics* **41**(3), 335–350.

Richard, A. (2006), Modèle formel pour les réseaux de régulation génétique et influence des circuits de rétroaction, PhD thesis, Evry-Val d'Essonne.

Richard, A. (2010), 'Negative circuits and sustained oscillations in asynchronous automata networks', *Advances in Applied Mathematics* **44**(4), 378–392.

Richard, A. & Comet, J.-P. (2007), 'Necessary conditions for multistationarity in discrete dynamical systems', *Discrete Applied Mathematics* **155**(18), 2403–2413.

Rizk, A., Batt, G., Fages, F. & Soliman, S. (2008), On a continuous degree of satisfaction of temporal logic formulae with applications to systems biology, *in* 'International Conference on Computational Methods in Systems Biology', Springer Berlin Heidelberg, pp. 251–268.

Rocca, A., Mobilia, N., Fanchon, É., Ribeiro, T., Trilling, L. & Inoue, K. (2014), Asp for construction and validation of regulatory biological networks, *in* L. F. del Cerro & K. Inoue, eds, 'Logical Modeling of Biological Systems', Wiley-ISTE, pp. 167–206.

Röhl, I. . A. (2006), 'De l'adn aux protéines'.
`http://www.cite-sciences.fr`

Samaga, R., Saez-Rodriguez, J., Alexopoulos, L. G., Sorger, P. K. & Klamt, S. (2009), 'The logic of egfr/erbb signaling : Theoretical properties and analysis of high-throughput data', *PLoS Computational Biology* **5**(8), e1000438.

Sanchez, L. & Thieffry, D. (2001), 'A logical analysis of the drosophila gap-gene system', *Journal of theoretical Biology* **211**(2), 115–141.

Schaefer, C. F., Anthony, K., Krupa, S., Buchoff, J., Day, M., Hannay, T. & Buetow, K. H. (2009), 'PID : the Pathway Interaction Database', *Nucleic Acids Research* **37**(suppl 1), D674–D679.

Schaffter, T., Marbach, D. & Floreano, D. (2011), 'Genenetweaver : in silico benchmark generation and performance profiling of network inference methods', *Bioinformatics* **27**(16), 2263–2270.

Schnoebelen, P. (2002), 'The complexity of temporal logic model checking.', *Advances in modal logic* **4**(393-436), 35.

Siebert, H. & Bockmayr, A. (2006), Incorporating time delays into the logical analysis of gene regulatory networks, *in* 'International Conference on Computational Methods in Systems Biology', Springer, pp. 169–183.

Silvescu, A. & Honavar, V. (2001), 'Temporal boolean network models of genetic networks and their inference from gene expression time series', *Complex Systems* **13**(1), 61–78.

Simao, E., Remy, E., Thieffry, D. & Chaouiya, C. (2005), 'Qualitative modelling of regulated metabolic pathways : application to the tryptophan biosynthesis in e. coli', *Bioinformatics* **21**(suppl 2), ii190–ii196.

Snoussi, E. H. (1989), 'Qualitative dynamics of piecewise-linear differential equations : a discrete mapping approach', *Dynamics and Stability of Systems* **4**(3-4), 565–583.

Somogyi, R. & Greller, L. D. (2001), 'The dynamics of molecular networks : applications to therapeutic discovery', *Drug discovery today* **6**(24), 1267–1277.

Talikka, M., Boue, S. & Schlage, W. K. (2015), 'Causal biological network database : A comprehensive platform of causal biological network models focused on the pulmonary and vascular systems', *Computational Systems Toxicology* pp. 65–93.

Terfve, C. & Saez-Rodriguez, J. (2012), Modeling signaling networks using high-throughput phospho-proteomics, *in* 'Advances in Systems Biology', Springer, pp. 19–57.

Thieffry, D. & Thomas, R. (1995), 'Dynamical behaviour of biological regulatory networks—ii. immunity control in bacteriophage lambda', *Bulletin of Mathematical Biology* **57**(2), 277–297.

Thierry-Mieg, Y., Poitrenaud, D., Hamez, A. & Kordon, F. (2009), Hierarchical set decision diagrams and regular models, *in* S. Kowalewski & A. Philippou, eds, 'Tools and Algorithms for the Construction and Analysis of Systems', Vol. 5505 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 1–15.

Thomas, R. (1973), 'Boolean formalization of genetic control circuits', *Journal of Theoretical Biology* **42**(3), 563 – 585.

Thomas, R. (1978), 'Logical analysis of systems comprising feedback loops', *Journal of Theoretical Biology* **73**(4), 631–656.

Thomas, R. (1981), On the relation between the logical structure of systems and their ability to generate multiple steady states or sustained oscillations, *in* J. Della Dora, J. Demongeot & B. Lacolle, eds, 'Numerical Methods in the Study of Critical Phenomena', Vol. 9 of *Springer Series in Synergetics*, Springer Berlin Heidelberg, pp. 180–193.

Thomas, R. (1991), 'Regulatory networks seen as asynchronous automata : a logical description', *Journal of theoretical biology* **153**(1), 1–23.

Thomas, R. & d'Ari, R. (1990), *Biological feedback*, CRC press.

Videla, S., Guziolowski, C., Eduati, F., Thiele, S., Gebser, M., Nicolas, J., Saez-Rodriguez, J., Schaub, T. & Siegel, A. (2014), 'Learning boolean logic models of signaling networks with asp', *Theoretical Computer Science* .

Villaverde, A. F., Becker, K. & Banga, J. R. (2016), Premer : Parallel reverse engineering of biological networks with information theory, *in* 'International Conference on Computational Methods in Systems Biology', Springer, pp. 323–329.

Wuensche, A. (1998), Genomic regulation modeled as a network with basins of attraction, *in* R. B. Altman, A. K. Dunker, L. Hunter & T. E. Klien, eds, 'Pacific Symposium on Biocomputing', Vol. 3, World Scientific, pp. 89–102.

Yamamoto, Y., Rougny, A., Nabeshima, H., Inoue, K., Moriya, H., Froidevaux, C. & Iwanuma, K. (2014), Completing sbgn-af networks by logic-based hypothesis finding, *in* 'Formal Methods in Macro-Biology', Springer, pp. 165–179.

Yu, N., Seo, J., Rho, K., Jang, Y., Park, J., Kim, W. K. & Lee, S. (2012), 'hipathdb : a human-integrated pathway database with facile visualization', *Nucleic acids research* **40**(D1), D797–D802.

Zhang, L., Jansen, D. N., Nielson, F. & Hermanns, H. (2011), Automata-based csl model checking, *in* 'Automata, Languages and Programming', Springer, pp. 271–282.

Zhang, S.-Q., Hayashida, M., Akutsu, T., Ching, W.-K. & Ng, M. K. (2007), 'Algorithms for finding small attractors in boolean networks', *EURASIP Journal on Bioinformatics and Systems Biology* **2007**(1), 1–13.

Zhang, Z.-Y., Horimoto, K. & Liu, Z. (2008), 'Time series segmentation for gene regulatory process with time-window-extension'.

Zhao, Z., Liu, C.-W., Wang, C.-Y. & Qian, W. (2014), Bdd-based synthesis of reconfigurable single-electron transistor arrays, *in* 'Proceedings of the 2014 IEEE/ACM International Conference on Computer-Aided Design', IEEE Press, pp. 47–54.

UNIVERSITE BRETAGNE LOIRE

CENTRALE NANTES

Thèse de Doctorat

Emna BEN ABDALLAH

Étude de la dynamique des réseaux biologiques : apprentissage des modèles, intégration des données temporelles et analyse formelle des propriétés dynamiques

Study of the dynamics of biological networks: learning models, time data integration and model checking analysis

Résumé

Au cours des dernières décennies, l'émergence d'une large gamme de nouvelles technologies a permis de produire une quantité massive de données biologiques (génomique, protéomique...). Ainsi, une grande quantité de données de séries temporelles est maintenant élaborée tous les jours. Nouvellement produites, ces données peuvent nous fournir des nouvelles interprétations sur le comportement des Systèmes Biologiques (SB). Cela conduit alors à des développements considérables dans le domaine de la *bioinformatique* qui peuvent tirer profit de ces données. Ceci justifie notre motivation pour le développement de méthodes efficaces qui exploitent ces données pour l'apprentissage des Réseaux de Régulation Biologique (RRB) modélisant les SB. Nous introduisons alors, dans cette thèse, une nouvelle approche qui infère des RRB à partir des données de séries temporelles. Les RRB appris sont présentés avec un nouveau formalisme, introduit dans cette thèse, appelé " *réseau d'automates avec le temps*" (T-AN). Ce dernier assure le raffinement de la dynamique des RRB, modélisés avec le formalisme des réseaux d'automates (AN), grâce à l'intégration d'un paramètre temporel (délai) dans les transitions locales des automates. Cet enrichissement permet de paramétrer les transitions entre les états locaux des automates et aussi entre les états globaux du réseau.
À posteriori de l'apprentissage des RRB, et dans le but d'avoir une meilleure compréhension de la nature du fonctionnement des SB, nous procédons à l'analyse formelle de la dynamique des RRB. Nous introduisons alors des méthodes logiques originales (développées en *Answer Set Programming*) pour déchiffrer l'énorme complexité de la dynamique des SB. Les propriétés dynamiques étudiées sont : l'identification des attracteurs (ensemble d'états globaux terminaux dont le réseau ne peut plus s'échapper) et la vérification de la propriété d'atteignabilité d'un objectif (un ensemble de composants) à partir d'un état global initial du réseau.

Mots clés

réseaux de régulation biologique, systèmes complexes, réseaux d'automates, réseaux d'automates avec le temps, apprentissage des modèles, analyse formelle, atteignabilité, attracteurs, Answer Set Programming

Abstract

Over the last few decades, the emergence of a wide range of new technologies has produced a massive amount of biological data (genomics, proteomics...). Thus, a very large amount of *time series data* is now produced every day. The newly produced data can give us new ideas about the behavior of biological systems. This leads to considerable developments in the field of *bioinformatics* that could benefit from these enormous data. This justifies the motivation to develop efficient methods for *learning* Biological Regulatory Networks (BRN) modeling a biological system from its time series data. Then, in order to understand the nature of system functions, we study, in this thesis, the dynamics of their BRN models. Indeed, we focus on developing original and scalable logical methods (implemented in Answer Set Programming) to deciphering the emerging complexity of dynamics of biological systems.
The main contributions of this thesis are enumerated in the following. (i) Refining the dynamics of the BRN, modeling with the automata Network (AN) formalism, by integrating a temporal parameter (*delay*) in the local transitions of the automata. We call the extended formalism a *Timed Automata Network* (T-AN). This integration allows the parametrization of the transitions between each automata local states as well as between the network global states. (ii) Learning BRNs modeling biological systems from their time series data. (iii) Model checking of discrete dynamical properties of BRN (modeling with AN and T-AN) by dynamical formal analysis : *attractors* identification (minimal trap domains from which the network cannot escape) and *reachability* verification of an objective from a network global initial state.

Key Words

Biological Regulatory Networks, complex systems, automata networks, timed automata networks, learning models, formal analysis, reachability, attractors, Answer Set Programming