Simulation développements et discussion

3.1 Simulation et complexité

La complexité est une théorie qui repose sur la classification de problèmes en différentes classes. Le type de problème le plus commun et le plus étudié est le problème de décision. Un tel problème pose une question à laquelle on doit répondre soit par "oui", soit par "non". L'ensemble des questions que l'on peut poser sont les instances du problème, qui sont dites positives lorsque la réponse est "oui", et négatives lorsque la réponse est "non". En général, les instances sont considérées comme des nombres plutôt que des phrases ; il est simple de voir que l'on peut toujours transformer l'un en l'autre.

Nous faisons dans ce chapitre la comparaison entre la réduction et la simulation. Ces deux opérations visent à démontrer que l'on peut résoudre un objet en en utilisant un autre, en permettant une mesure de complexité. Nous sommes intéressés à pousser l'analogie le plus loin possible, pour comprendre quelles structures sont conservées dans la traduction, et lesquelles sont transformées. Nous supposons la définition de simulation développée en section 2.3.2 car, bien qu'elle soit incomplète, elle reste la définition la plus satisfaisante. Pour rappel, la définition de réduction est donnée en section 1.8, accompagnée des autres définitions de base liées à la théorie de la complexité.

3.1.1 Problèmes et systèmes de transition d'états

La première différence entre réduction et simulation est que les objets sur lesquels ces notions opèrent sont fondamentalement différents : les problèmes disposent toujours d'un nombre infini d'instances, et la difficulté de ces problèmes repose uniquement sur la répartition des réponses positives à ces instances. Un système de transition d'états est bien plus libre dans sa structure ; un tel système peut à priori être de toute taille, même finie, et peut inclure des chemins infinis lorsqu'il est infini.

Nous observons ici qu'il existe une façon simple de transformer tout problème de décision en système de transition d'états.

Définition 12. *Soit A un problème de décision disposant d'un ensemble d'instances I ; la transformation de ce problème est un système de transition d'états que nous notons* $\mathrm{STE}(A)$ *qui dispose des états $I \cup \{o, p\}$ où o correspond à la réponse "oui" et p correspond*

à la réponse "non". L'ensemble des actions Λ de ce système est l'ensemble des nombres entiers, que l'on encode en unaire. Pour tout instance i du problème A, on a $i \xrightarrow{n} o$ si et seulement si i est une instance positive de taille n, et $i \xrightarrow{n} p$ si et seulement si i est une instance négative de taille n.

Exemple 27. *On considère SAT le problème de décision qui repose sur la satisfiabilité d'une formule booléenne. Son ensemble d'instances I est l'ensemble des formules booléennes, qui sont définies en section 1.5. Le système de transition d'états $\mathtt{STE}(A_{SAT})$ est composé des états $I \cup \{o, p\}$, soit l'ensemble de toutes les formules booléennes, et les états spéciaux o et p. Pour toute instance i encodable en n bits, ce système admet $i \xrightarrow{n} o$ si i est une formule satisfiable. Si i est insatisfiable, alors le système admet $i \xrightarrow{n} p$.*

Nous utilisons les actions de ces systèmes pour transmettre à la fonction g la taille de l'instance. Cela permet à la fonction g de décider du temps nécessaire à attendre pour résoudre une instance donnée. Cette façon de traduire un problème de décision peut paraître artificielle, et témoigne des différences entre problèmes de décision et systèmes de transition d'états. Pour traduire les problèmes en systèmes, certaines technicités sont nécessaires.

3.1.2 Classes de simulation

La définition des classes de complexité repose sur la résolution de problèmes par des machines limitées en temps et en espace. Nous construisons une analogie basée sur la simulation en remplaçant la condition de résoudre le problème par une machine par la condition que le système étudié soit simulable par la même machine disposant des mêmes restrictions.

Pour permettre cette traduction, nous décrivons les systèmes de transition d'états qui correspondent aux machines de Turing, dont la définition est donnée en section 1.6.

Définition 13. *Pour $T = (S, \Lambda, s_0, \delta, F)$ une machine de Turing déterministe, $\mathtt{STE}(T)$ est le système de transition d'états qui admet les états $S \times \Lambda^{\mathbb{Z}} \times \mathbb{Z}$, l'ensemble d'actions $\{a\}$ et la relation de transition $\rightarrow_T$ telle que, pour toute paire d'états (s, R, z), (s', R', z'), $(s, R, z) \rightarrow_T (s', R', z')$ si et seulement si $\delta(s, R_z) = (s', R'_{z'}, z' - z)$ et $R_k = R'_k$ pour tout $k \neq z$.*

Pour (s, R, z) un état de $\mathtt{STE}(T)$, R représente l'état du ruban, s l'état de la machine et z la position de la tête de lecture. Cette définition est également généralisable au cas non déterministe.

Définition 14. *Pour $T = (S, \Lambda, s_0, \Delta, F)$ une machine de Turing non déterministe, $\mathtt{STE}(T)$ est le système de transition d'états qui admet les états $S \times \Lambda^{\mathbb{Z}} \times \mathbb{Z}$, l'ensemble d'actions $\{a\}$ et la relation de transition $\rightarrow_T$ telle que, pour toute paire d'états (s, R, z), (s', R', z'), $(s, R, z) \rightarrow_T (s', R', z')$ si et seulement si $(s, R_z, s', R'_{z'}, z' - z) \in \Delta$ et $R_k = R'_k$ pour tout $k \neq z$.*

Nous faisons l'observation importante que, telle que décrite dans ces définitions, chaque configuration de ces systèmes de transition d'états contient l'information d'un vecteur bi-infini dans $\Lambda^{\mathbb{Z}}$. Cette observation est problématique puisque ce vecteur passe par définition en entrée et sortie des fonctions qui composent une simulation polynomiale. Si ces entrées et sorties sont infinies, la question d'une mesure de complexité devient inapplicable. Pour contourner ce problème, nous considérons que ce ruban peut être alternativement défini comme la liste des positions dont l'état n'est pas 0, pour 0 un état arbitraire de Λ. Cet encodage permet de définir la simulation d'un modèle à description finie par une machine de Turing, et de mesurer cette simulation en complexité sous l'hypothèse que les configurations employées dans cette simulation comportent toutes un nombre fini de positions dont la valeur n'est pas 0. Il n'est cependant pas difficile de voir que cette hypothèse est conservée dans l'ensemble des simulations considérées dans ce chapitre.

Ces définitions de systèmes de transitions d'états permettent la définition intuitive des classes $\mathsf{P}_{\preccurlyeq}$ et $\mathsf{NP}_{\preccurlyeq}$ suivante.

Définition 15 (La classe $\mathsf{P}_{\preccurlyeq}$)**.** *La* classe $\mathsf{P}_{\preccurlyeq}$ *est l'ensemble des systèmes de transition d'états* $(S, \Lambda, \rightarrow)$ *pour lesquels il existe une machine de Turing déterministe* T *telle que* $(S, \Lambda, \rightarrow) \preccurlyeq_L \mathtt{STE}(T)$.

Définition 16 (La classe $\mathsf{NP}_{\preccurlyeq}$)**.** *La* classe $\mathsf{NP}_{\preccurlyeq}$ *est l'ensemble des systèmes de transition d'états* $(S, \Lambda, \rightarrow)$ *pour lesquels il existe une machine de Turing non déterministe* T *telle que* $(S, \Lambda, \rightarrow) \preccurlyeq_P \mathtt{STE}(T)$.

Naturellement, la définition de ces classes permet de vérifier que la classe $\mathsf{P}_{\preccurlyeq}$ est incluse dans la classe $\mathsf{NP}_{\preccurlyeq}$.

Propriété 5.

$$P_{\preccurlyeq} \subseteq NP_{\preccurlyeq}$$

Démonstration. On observe que toute machine déterministe $T = (S, \Lambda, s_0, \delta, F)$ est également la machine non déterminisite $(S, \Lambda, s_0, \Delta, F)$ telle que $(s, x, s', y, p) \in \Delta$ si et seulement si $\delta(s, x) = (s', y, p)$. De plus, toute simulation en espace logarithmique est également une simulation en temps polynomial. On obtient donc que tout système de transition d'états inclus dans $\mathsf{P}_{\preccurlyeq}$ est également inclus dans $\mathsf{NP}_{\preccurlyeq}$. □

La comparaison entre ces classes de simulation et les classes de complexité ne s'arrête pas là. Nous montrons qu'un problème A est dans P si et seulement si sa traduction $\mathtt{STE}(A)$ est dans $\mathsf{P}_{\preccurlyeq}$.

Propriété 6. *Soit A un problème de décision. Si $A \in P$, $\mathtt{STE}(A) \in P_{\preccurlyeq}$.*

Démonstration. Par hypothèse, il existe une machine de Turing T qui résout le problème A. Pour prouver la simulation, nous définissons la fonction h comme définie sur le domaine des configurations d'une machine de Turing qui est dans un état initial (et encode une instance de notre problème), ou final. Ainsi $h(s')$ vaudra i si la

configuration s' est dans l'état initial et contient l'instance i du problème A. Sinon, $h(s')$ vaudra o si la machine est dans un état final acceptant, et p si la machine est dans un état final de refus. Cette fonction est calculable en espace logarithmique ; d'abord, dans le cas d'un état final , la fonction est calculable en temps constant. Dans le cas d'une configuration initiale, on considère que h répond simplement l'instance rédigée dans le ruban de la machine dans son état initial, ce qui est clairement calculable en espace logarithmique. Cela fonctionne car on considère que toute instance mal formée du problème A est une instance négative du problème A. Grâce à cette hypothèse, h admet toute configuration initiale de la machine dans son domaine, et chaque instance de A est transformée en exactement une configuration initiale. Autrement dit, la charge de la preuve que i est une instance bien formée est laissée à la machine de Turing T plutôt qu'à h.

Par hypothèse que $A \in \mathsf{P}$, il existe un polynôme f qui pour toute taille d'instance n borne le nombre de mises à jour nécessaire pour décider toute instance de taille n grâce la machine qui la résout. On suppose ici que la machine T résout toute instance de taille n en exactement $f(n)$ étapes, puis s'arrête. La fonction g prend en entrée un nombre n égal à la taille de l'instance encodé en unaire, et calcule une séquence définie comme la répétition $a^{f(n)}$. Grâce au fait que n est encodé en unaire, ce processus peut-être opéré en espace logarithmique : on compte d'abord la taille de l'entrée en binaire, puis on applique le polynôme f. On peut déduire que le bit numéro k de la sortie est un a si et seulement si $k < f(n)$.

Il s'en suit que g et h sont toutes les deux calculables en espace logarithmique. On note également que h est surjective, car toute instance correspond à au moins une configuration initiale, et les réponses o et p sont obtenues depuis toute configuration finale.

Soient S, Λ les ensembles d'états et d'actions de $\mathtt{STE}(A)$, et S' l'ensemble des états de $\mathtt{STE}(T)$.

Supposons $s, r \in S$, $a \in \Lambda$ et $s' \in \mathrm{dom}(h)$, tels que $h(s') = s$ et $s \xrightarrow{a} r$. L'état s est nécessairement une instance i du problème A. Alors s' est une configuration initale de machine de Turing qui encode l'instance i, et $a = |i|$. Il existe donc $r' \in \mathrm{dom}(h)$ la configuration qui correspond au résultat du calcul de la machine qui prend i en entrée, telle que $h(r') = r$ (le résultat de la machine est cohérent avec la réponse à l'instance), et $s' \xrightarrow{g(a)} {}' r'$ (par hypothèse la machine trouve ce résultat après exactement $|g(|i|)|$ étapes de temps). La première clause de la simulation est donc vérifiée.

Supposons maintenant $s \in S$, $a \in \Lambda$, et $s', r' \in \mathrm{dom}(h)$ tels que $h(s') = s$ et $s' \xrightarrow{g(a)} {}' r'$. On sait de nouveau que s est une instance i de A, car dans le cas contraire s' serait une configuration finale de la machine, qui n'a pas de successeur. Alors s' est une configuration initiale qui encode cette instance, et r' est donc nécessairement une configuration finale. Par l'hypothèse que la machine résout l'instance en exactement $f(|i|)$ étapes et donc $a = |i|$. On obtient donc que $s \xrightarrow{a} h(r')$, ce qui prouve la seconde clause de la simulation.

□

Propriété 7. *Soit A un problème de décision. Si* $\mathtt{STE}(A) \in P_{\preccurlyeq}$*, alors* $A \in P$.

Démonstration. Par hypothèse, pour toute instance i du problème A, il existe une configuration s' d'une machine de Turing T donnée telle que $h(s') = i$. Après $|g(|i|)| = f(n)$ mises à jour, pour f un polynôme, on obtient une configuration de machine de Turing r' telle que $r' \in \mathrm{dom}(h)$ et $h(r')$ est la réponse à l'instance i. Il est à noter que la configuration r' n'a aucune garantie d'être finale, pas plus que s' n'a de garantie d'être initiale. Cependant comme par hypothèse $h(r')$ peut être calculée en espace logarithmique, nous pouvons construire T' une machine qui se comporte de façon identique à la machine T, à l'exception qu'une fois à la configuration r', T' calcule $h(r')$ et converge vers l'état d'acceptation ou de refus correspondant. Cette machine T' résout alors i en temps polynomial en démarrant depuis la configuration s' modifiée pour que la machine soit alors dans un état initial. □

Il est intéressant qu'un problème P-difficile ne se traduise donc pas en un système $\mathrm{P}_{\preccurlyeq}$-difficile, en supposant que cette dernière notation signifie que le système en question est capable de simuler en espace logarithmique tout autre système dans $\mathrm{P}_{\preccurlyeq}$. Une instance donnée d'un problème de décision n'est capable de prédire l'état d'une machine de Turing qu'après un nombre fini de mises à jour, et ne pourra donc pas simuler les détails d'une exécution potentiellement sans fin. Il existe cependant bien des modèles $\mathrm{P}_{\preccurlyeq}$-difficiles : les modèles $\mathtt{STE}(T)$, pour T toute machine de Turing universelle, en sont des exemples qui découlent directement de la définition de $\mathrm{P}_{\preccurlyeq}$. L'intuition nous invite même à formuler la conjecture suivante. Ici, *efficacement Turing universel* signifie capable de simuler toute machine de Turing de façon à ce que le décodage d'une configuration (calcul de l'état, des symboles inscrits sur le ruban, et de la position de la tête) soit réalisable en espace logarithmique.

Conjecture 1. *L'ensemble des systèmes efficacement Turing universels est l'ensemble des systèmes $P_{\preccurlyeq}$-difficile.*

Essayons maintenant de produire les même résultats pour les relations entre les classes NP, coNP et $\mathrm{NP}_{\preccurlyeq}$. Les problèmes de la classe NP peuvent être décrits de la façon suivante : ils peuvent être résolus en temps polynomial par une machine de Turing non déterministe, c'est-à-dire en particulier que si l'instance étudiée est positive, alors il existe une branche de taille polynomiale dans l'exécution de la machine qui retourne la réponse "oui". Les problèmes de la classe coNP sont similaires, et si une instance étudiée d'un tel problème est négative, alors il existe au moins une branche de taille polynomiale dans l'exécution de la machine qui répond "non". Intuitivement, une instance positive d'un problème dans NP se vérifie facilement dès que l'on dispose du bon indice, et une instance négative d'un problème dans coNP se démontre facilement fausse dès que l'on dispose d'un bon contre-exemple.

La distinction entre ces deux classes est cohérente dans la théorie de la complexité, bien que leur égalité reste un problème ouvert, car la notion de réduction ne permet pas l'inversion de la réponse aux instances. Tout problème dans NP dispose d'un dual dans coNP, où les indices deviennent des contre-exemples. Les problèmes

NP-complets n'admettent pas de réduction polynomiale connue vers leurs duaux respectifs, et l'existence d'une telle réduction impliquerait que NP = coNP. Au meilleur de nos connaissances actuelles, l'égalité entre ces classes est un problème ouvert.

Dans le cas de la simulation telle que nous la définissons, il n'existe pas de convention pour l'acceptation ou le refus d'une instance. Cela semble logique ; dans le cas le plus général, les modèles de calcul n'ont pas la vocation de résoudre des problèmes, où même la vocation de faire quoi que ce soit en particulier. Un automate cellulaire qui évolue sur une configuration infinie va évoluer sur une durée de temps infinie, et aucune convention de résultat d'un calcul ne ferait sens en dehors d'un contexte très précis. Nous faisons même la remarque que c'est le rôle d'une définition générale de la simulation de permettre l'application d'un tel contexte sur le calcul, et en particulier de définir les conventions qui définissent la validité ou l'invalidité d'une instance par exemple dans le cas d'une simulation d'un problème de décision par la règle 110, qui est un modèle de calcul universel. De cette impossibilité de distinguer les conventions de validation et de refus ressort le fait suivant : il n'existe pas de traduction de NP et coNP qui permette de les distinguer dans le contexte de la simulation.

Le problème est encore plus profond. La définition des classes NP et coNP permet de définir la façon dont les problèmes concernés doivent être résolus. Ainsi, l'instance d'un problème dans NP est positive s'il existe une branche de calcul qui répond "oui" dans la machine de Turing correspondante, ce qui correspond à notre définition de simulation. Cependant, notre définition de simulation, qui ne traite pas le "oui" de façon différente du "non", va également interpréter une instance comme négative s'il existe une branche de calcul qui répond "non".

Pour contourner ce problème, nous proposons de conserver $\mathsf{NP}_{\preccurlyeq}$ comme une classe englobant les traductions des problèmes de NP et coNP. Nous faisons la distinction par la façon de traduire le problème de décision concerné en un système de transition d'états.

Définition 17 (Système positif d'un problème de décision)**.** *Soit A un problème de décision, avec un ensemble d'instances I. Le système de transition d'états positif de A, noté $\mathtt{STE}^+(A)$, est défini sur les nœuds $I \cup \{o\}$. Son ensemble d'actions est l'ensemble des entiers encodés en unaire. Pour toute instance $i \in I$, ce système admet $i \xrightarrow{n} o$ où n est la taille de l'instance i.*

Définition 18 (Système négatif d'un problème de décision)**.** *Soit A un problème de décision, avec un ensemble d'instances I. Le système de transition d'états négatif de A, noté $\mathtt{STE}^-(A)$, est défini sur les nœuds $I \cup \{p\}$. Son ensemble d'actions est l'ensemble des entiers encodés en unaire. Pour toute instance $i \in I$, ce système admet $i \xrightarrow{n} p$ où n est la taille de l'instance i.*

Cette redéfinition de la traduction d'un problème de décision nous permet de nous concentrer sur les instances qui sont considérées importantes pour les définitions des classes qui nous intéressent.

Propriété 8.

$$A \in NP \iff \mathtt{STE}^{+}(A) \in NP_{\preccurlyeq}$$
$$A \in coNP \iff \mathtt{STE}^{-}(A) \in NP_{\preccurlyeq}$$

Démonstration. Pour prouver que $A \in \mathsf{NP} \implies \mathtt{STE}^{+}(A) \in \mathsf{NP}_{\preccurlyeq}$ et $A \in \mathsf{coNP} \implies \mathtt{STE}^{-}(A) \in \mathsf{NP}_{\preccurlyeq}$, nous observons une simulation identique à la simulation définie dans la preuve de la propriété 6. Le résultat découle du fait que la définition précise des $\mathtt{STE}^{+}(A)$ et $\mathtt{STE}^{-}(A)$ requiert que la simulation ne couvre que les cas qui sont garantis par les hypothèses $A \in \mathsf{NP}$ et $A \in \mathsf{coNP}$ respectivement. Pour prouver que $\mathtt{STE}^{+}(A) \in \mathsf{NP}_{\preccurlyeq} \implies A \in \mathsf{NP}$ et $\mathtt{STE}^{-}(A) \in \mathsf{NP}_{\preccurlyeq} \implies A \in \mathsf{coNP}$, nous observons un ensemble d'arguments similaires à la preuve de la propriété 7. □

La méthode employée pour obtenir ce résultat est questionnable : il ne serait pas raisonnable de définir une façon différente de perçevoir les problèmes de décision à chaque nouvelle classe que nous voudrions reproduire. Cette solution *ad hoc* est en vérité le témoin d'une différence importante entre la théorie de la complexité et l'extension que nous tentons. La théorie de la complexité fait la distinction entre la réduction, qui permet la comparaison entre problèmes, et la résolution d'un problème par une machine, qui est fondamentale pour la définition des classes de complexité. Dans le cadre que nous étudions, ces deux notions sont interprétées par la simulation. L'élégance éventuelle de cette manœuvre retire la liberté de définir les classes par la façon spécifique dont la machine résout le problème. Pour en reproduire la structure, nous sommes contraints de reproduire cette spécificité là où cela est possible, spécifiquement dans la traduction des problèmes de décision que nous admettons.

Il est important de rappeler que les notions de réduction et de simulation restent fondamentalement différentes car définies sur des objets différents. Si nous trouvons encourageant de trouver des similarités entre les structures qui sont obtenues par la théorie de la complexité et la simulation, il est non moins encourageant de comprendre les structures des classes de simulation pour ce qu'elles sont, et en particulier en la façon dont elles divergent de notre compréhension. Car, bien que ce manuscrit ne se concentre que sur l'adaptation d'un fragment réduit de la théorie de la complexité dans le contexte de la simulation comme d'un gage de cohérence, la simulation permet l'inclusion de bien plus de formes de calcul, qui dépassent la cadre habituel de la complexité.

3.2 Intuition générale de la simulation

Lorsque nous étudions des modèles mathématiques, nous discutons souvent des intuitions qui se cachent derrière les définitions. La simulation n'est pas en reste ; cette notion, qui a été formellement développée indépendamment dans de nombreux domaines, nous encourage à la généraliser, par l'intuition que si l'ensemble des domaines concernés ont employé ce terme pour parler de choses indépendantes, c'est

qu'il existe une perception commune de ce qu'est la simulation. Si cela est avéré, cela nous encouragerait à poursuivre cette perception dans notre recherche d'une définition générale. Quelle est cette perception ? Pour conclure notre travail sur la simulation, nous proposons dans cette section une forme de discussion sur les idées qui entourent la simulation. Nous essayons, informellement, de comprendre ce qui lie la notion de simulation à la notion informelle de calcul ; et nous faisons des liens avec l'idée d'émergence.

Ce discours informel ne cherche pas à permettre l'élaboration d'un quelconque théorème, ou de tout autre résultat formel mesurable. Il cherche plutôt à organiser un ensemble d'images afin de clarifier notre propre perception du terme de simulation, et peut-être aussi d'élaborer des outils propres à l'éducation et la vulgarisation des notions employées.

3.2.1 La simulation et la notion de calcul

Quelle est l'intuition commune derrière la simulation ? Dans la pratique, la simulation permet de montrer qu'un modèle peut calculer tout ce qu'un autre modèle calcule. Il semble donc inévitable que si l'on veut parler de simulation, on doit également parler de calcul.

Le calcul est une notion qui dispose d'énormément d'exemples formels ; ils sont aussi nombreux que les modèles de calcul qui en portent le nom, mais aussi les très nombreux algorithmes et machines concrètes qui parcourent les domaines appliqués de l'informatique. Il est pourtant difficile d'extraire de tous ces exemples l'ensemble des critères formels qui établissent que quelque chose "calcule". Le travail qui semble se rapprocher le plus de cette formalisation est la thèse de Church-Turing, qui stipule l'existence de modèles aux calculs universels. Cependant, si elle nous donne un exemple de modèle universel, la thèse de Church-Turing ne permet pas de trouver les critères qui font qu'un modèle est universel.

Nous posons la notion du calcul de la façon suivante : un calcul est un processus qui permet la résolution d'un problème. Ainsi, si un automate cellulaire élémentaire (voir section 1.7) ne possède pas la fonction première de résoudre un problème pratique, la littérature qui s'y intéresse mesure la capacité de calcul des différentes règles élémentaires par leur capacité à résoudre des problèmes. Par le biais de la simulation illustrée en figure 2.1, on exprime que la règle 134 possède une capacité de calcul au moins aussi puissante que la règle 184 ; par la simulation développée par (Cook 2004), on sait que la règle 110 est capable de résoudre tout problème qu'une machine de Turing peut résoudre et, par conséquent, que son calcul est au moins aussi puissant que ces dernières.

Quelle est la fonction de la simulation dans le calcul ? La mention de simulation dans le précédent paragraphe met en avant la fonction première de la simulation en pratique : prouver la capacité de calcul d'un modèle en se servant du point de référence d'un autre modèle. Peut-on trouver une description plus absolue ?

Fondamentalement, il n'existe pas de problème sans intelligence pour se le poser.

On en vient donc au rôle de l'observateur dans la notion de calcul, et plus précisément, la façon dont le rôle de l'observateur sublime celui de la simulation dans notre relation humaine au calcul.

3.2.2 La simulation, outil pratique de traduction

Pour guider le développement de cette idée, nous proposons la série d'images suivantes.

Alice dispose d'un problème en tête. Il s'agit d'un problème arithmétique. Pour le résoudre, Alice rédige les équations pertinentes sur le papier et trouve la solution.

Bob dipose d'un problème arithmétique, et pour le résoudre dispose d'une calculatrice. Bob entre le problème dans la calculatrice et la calculatrice lui donne une réponse, que Bob recopie.

Charlie est abandonné sur une planète inconnue, et souhaite résoudre un problème arithmétique. Charlie est devant une machine extra-terrestre inconnue. Fort heureusement, Charlie dispose également d'un manuel d'utilisation, qui permet de se servir de cette machine inconnue comme d'une calculatrice. Charlie traduit son problème en quelque chose que la machine peut comprendre, puis interprète le résultat en quelque chose que lui même comprend, le tout grâce au manuel.

Dans ces trois histoires, chacun de nos acteurs effectue un calcul ; dans quelle histoire a-t-on un exemple de simulation ? L'histoire de Charlie est construite pour contenir la simulation la plus explicite. C'est le manuel d'instruction qui sert de preuve explicite que la machine extra-terrestre est capable du même calcul qu'une calculatrice. Est-ce la seule simulation que l'on peut extraire de ces histoires ?

En plus des machines concrètes décrites dans ces histoires – la calculatrice, la machine extra-terrestre – Alice, Bob et Charlie disposent d'un problème d'arithmétique qui réside dans leur tête. Ce problème peut être résolu ; et par notre définition de calcul, tout ce qui permet de le résoudre est alors un objet qui calcule. Cependant, nous estimons que ce problème est aussi lui-même un objet qui calcule car, à l'instar des problèmes de décision mentionnés en section 3.1, il est capable de prendre part dans une réduction avec d'autres problèmes, ou d'être simulé par des modèles. Les histoires d'Alice et Bob sont donc également une histoire de simulation, où la résolution de leur problème présuppose une simulation de l'ensemble des problèmes arithmétiques par les équations et la calculatrice respectivement. Cela est également observable dans l'histoire de Charlie, qui décrit finalement une simulation en deux étapes. Bien que Charlie ne dispose pas de calculatrice, mais seulement de la machine extra-terrestre, le manuel d'utilisation lui permet d'utiliser la complexité de la machine extra-terrestre comme d'une calculatrice : le manuel d'utilisation constitue en vérité une preuve de simulation d'une calculatrice par la machine extra-terrestre. En plus de cette simulation, Charlie utilise cette calculatrice virtuelle pour résoudre son problème. Et comme pour Alice et Bob, cela constitue en soi une simulation, qui est la seconde étape de la résolution du problème de Charlie.

Nous disons donc la chose suivante : dans tout emploi du calcul par un observateur

pour résoudre un problème, il réside une simulation parfois implicite qui repose sur la distance intrinsèque entre le problème posé et la machine qui, par la simulation, le résout. Par cette proposition, la simulation devient plus qu'un outil permettant la démonstration qu'un modèle est aussi expressif qu'un autre, mais également une condition nécessaire à l'emploi d'une machine pour toute tâche qui nécessite une interprétation, même triviale, pour sa résolution. Dans l'histoire de Bob, par exemple, l'utilisation du clavier de la calculatrice et la lecture de son écran sont des étapes nécessaires d'une interprétation simple de son calcul.

Que signifie cette observation ? Principalement que la simulation dépasse intuitivement le cadre spécifique des modèles de calcul naturel, et nous parle plus largement de notre rapport au calcul en général, qui est fondamental subjectif.

Pour mettre en avant ce dernier point, considérons de nouveau l'histoire de Charlie. Supposons que, dans une variante de cette histoire, Charlie ne dispose pas de ce manuel d'utilisation qui lui permet l'utilisation de la machine extra-terrestre. Maintenant, du point de vue de Charlie, la machine n'est plus capable de résoudre son problème d'arithmétique. Charlie pourra observer que le comportement de cette machine est complexe, mais en l'absence d'une façon concrète de faire sens de cette complexité, peut-on se convaincre que cette machine calcule ? Cette situation est analogue à beaucoup de positions de la littérature sur des modèles de calcul dont la capacité de calcul est supposée mais pas démontrée, comme l'automate cellulaire élémentaire 54 par exemple.

Cette histoire nous invite à réfléchir à l'espace entre l'observateur et la machine. Cet espace implique un travail de l'observateur pour interpréter sa machine, travail que nous nommons simulation. Si la machine extra-terrestre dispose d'une capacité intrinsèque à résoudre des problèmes, la simulation est le procédé qui permet de reconnaître et d'exploiter cette capacité. La simulation semble bien se définir de façon conjointe au calcul ; la simulation est la partie du calcul qui implique la traduction d'énoncés ou de solutions d'un modèle vers un autre.

3.2.3 Simulation et émergence

Dans l'histoire de Charlie, nous touchons à une idée très intéressante : l'idée que sans avoir la clé pour comprendre le calcul d'une machine complexe, nous sommes tout de même capables d'en percevoir superficiellement la complexité. En faisant l'observation d'un modèle tel que l'automate cellulaire élémentaire 54, nous faisons le constat que des comportements complexes émergent.

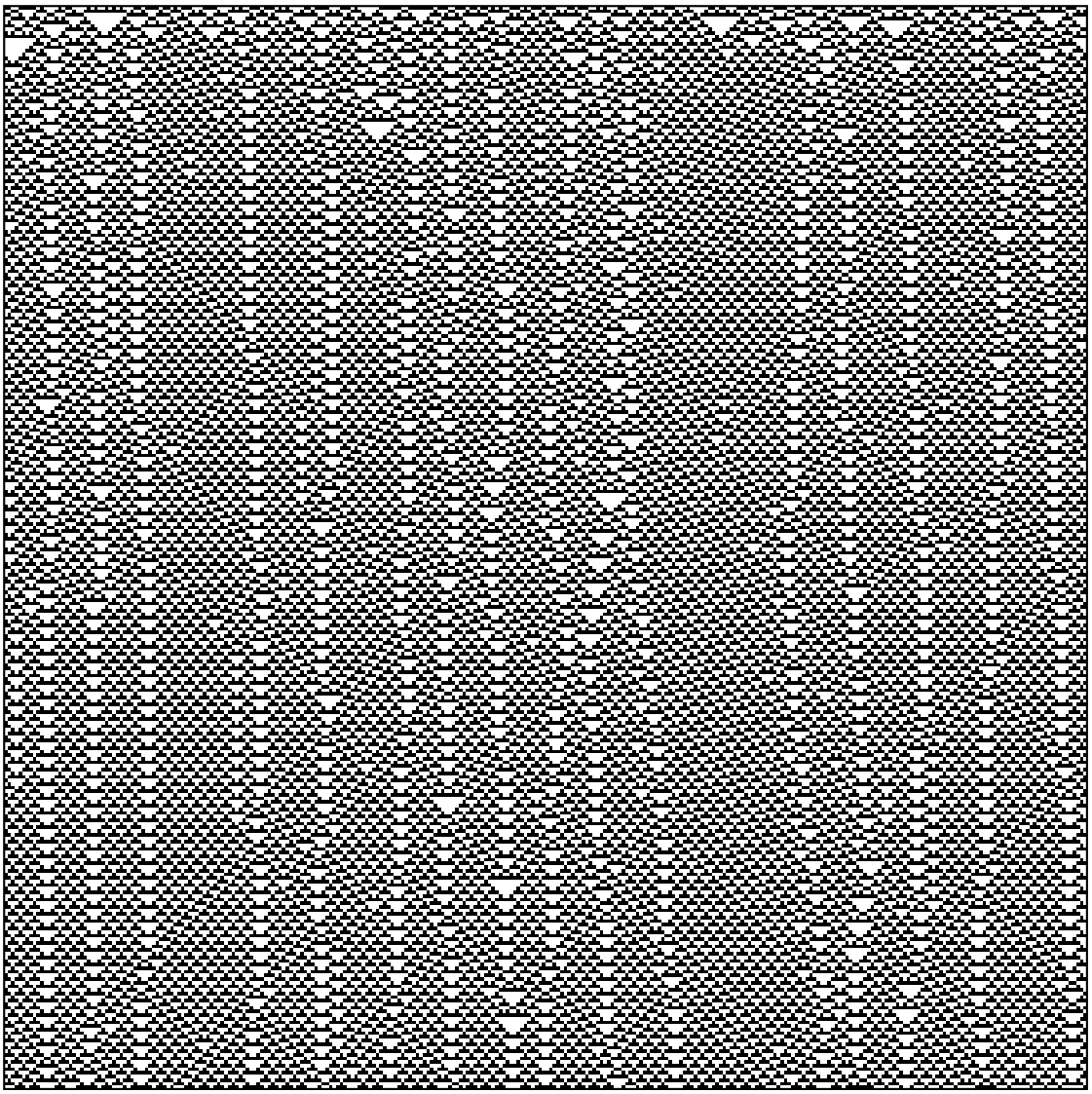

FIGURE 3.1 – Représentation d'une exécution de la règle 54. Cette règle est connue pour ses comportements complexes. D'une configuration initiale aléatoire émergent des structures qui s'apparentent à des particules qui évoluent, se déplacent et interagissent.

Qu'est-ce que l'émergence ? C'est une notion qui, historiquement, repose sur l'observation que, bien que l'univers semble régi par des règles fixées dont nous connaissons certains détails, il existe des choses dans cet univers d'une complexité telle qu'il est (à cet instant du moins) impossible de les déduire de ces règles élémentaires ; l'existence de la conscience humaine, quelle que soit sa définition, est un exemple commun de phénomène pour lequel une description complète apparaît hors de portée. Ceci engendre notamment la perte du lien causal intuitif entre les règles fondamentales et

les phénomènes observés. On dit de ces phénomènes complexes qu'ils *émergent* dans le système.

La question de définir la nature précise de cette émergence est une question difficile (LEWES 1875 ; O'CONNOR 1994 ; CRUTCHFIELD 1994). Parmi les définitions les plus populaires de l'émergence, on citera deux exemples : l'*émergence forte* affirme que les formes qui émergent à un niveau supérieur, c'est-à-dire les phénomènes complexes, possèdent une influence sur les composantes du niveau inférieur, c'est-à-dire les éléments les plus simples du système. Cette perspective est rarement acceptée par le point de vue scientifique, car elle évoque des positions controversées telles que le vitalisme (l'idée que le vivant ne se réduit pas à des règles physico-chimiques). L'*émergence faible* affirme que, bien que les phénomènes émergents fassent preuve d'une complexité qui justifie leur étude par la science, ces phénomènes sont bel et bien le fruit des règles élémentaires du système. Cette position fait la supposition que si, à un moment donné, nous échouons à décrire un phénomène à partir des règles fondamentales, il s'agit avant tout de la faute de notre compréhension incomplète de la réalité.

Il existe des travaux qui discutent de la relation entre émergence et simulation (BOSCHETTI 2012 ; WILDMAN et SHULTS 2018), mais la simulation dont il est question ici n'est pas la simulation employée dans ce chapitre, il s'agit plutôt de la sorte de simulation qui permet, par exemple, de vérifier une théorie fondamentale de la physique sur de puissantes machines. Dans cette section, nous sommes intéressés à tisser les liens entre émergence et simulation, principalement en suivant l'idée que l'émergence se définit souvent comme dépendante de la perspective (CRUTCHFIELD 1994). Cette notion signifie que notre proportion à décrire un phénomène comme émergent dépend principalement de notre compréhension de ce dernier. Moins nous le comprenons, et plus il nous semble justifié de le percevoir comme émergent. Plus nous le comprenons, et plus il nous semblera juste de le décrire comme une conséquence entendue de l'application des règles fondamentales.

Nous invitons à repenser à l'histoire de Charlie et de sa machine extra-terrestre, et de la façon dont, pour Charlie, la capacité de calcul de cette machine dépend de sa compréhension de cette dernière. Lorsque cette compréhension est absente, Charlie voudra exprimer la complexité apparente de cette machine comme un comportement émergent. Le manuel d'utilisation lui confère le savoir qui lui est nécessaire pour faire partiellement sens de cette complexité ; à l'aide de ce manuel, la machine cryptique devient, entre autres, une calculatrice, dont le fonctionnement est intuitivement compris.

La simulation, incarnée dans cette histoire par le manuel d'utilisation, devient alors l'incarnation formelle de l'explication qui permet de réduire certains comportements à d'autres.

Lorsque nous observons un phénomène complexe incompris d'un système de calcul, les éléments développés dans cette section invitent à établir qu'il est raisonnable de décrire ce phénomène comme émergent. Il semble également clair que, dès que la perception d'une émergence est expliquée par une simulation, le phénomène se

range alors dans la catégorie de l'émergence faible. La simulation, par sa notion très générale, semble être le représentant parfait des procédures qui nous permettent de comprendre et d'expliquer ce phénomène en le réduisant à d'autres phénomènes mieux compris. Son rôle vis-à-vis du calcul est en vérité fondamentalement epistémologique, c'est-à-dire que toute connaissance d'un calcul est apportée par une simulation. Plutôt que de créer du calcul, la simulation découvre, explique et détaille le calcul d'un modèle en le comparant au calcul d'un autre modèle.

3.2.4 Conclusion

La simulation touche, grâce au domaine du calcul naturel, aux intuitions qui appartiennent à de très nombreuses sciences comme la biologie ou la physique. La position unique de cette notion très générale fait partie des raisons qui nous poussent à mieux cerner l'intuition qui régit ses très nombreuses apparitions dans les domaines du calcul naturel.

Sur le plan formel, la simulation est un outil largement utilisé, sous diverses formes, pour décrire la complexité – dans le sens large du terme – des très nombreux modèles du calcul naturel. Sur le plan intuitif, il s'agit d'une notion dont les racines sont profondes et touchent aux relations fondamentales entre l'observateur et le calcul. Ces deux aspects font de la question de la généralisation de la simulation un projet ambitieux et dont les enjeux sont majeurs. Bien que le travail présenté dans ce manuscrit pose bien plus de questions qu'il n'apporte de réponses, nous espérons transmettre l'importance de cette direction particulière et chercherons à la développer dans le futur.

4 Réseaux d'automates

4.1 Introduction

Les réseaux d'automates tiennent leur origine dans la théorie des automates à seuils, dont la motivation initiale était d'imiter les réseaux de neurones observés dans le vivant (McCulloch et Pitts 1943). Bien que cette perspective ait donné naissance à la théorie des automates (Kleene et Post 1954 ; Elspas 1959 ; Golomb et al. 1967) et que la perpective en direction de la cognition ait amené aujourd'hui au domaine productif des réseaux de neurones dans l'apprentissage automatique , les réseaux d'automates – définis comme une vision plus large des automates à seuil et des réseaux de neurones – restent un sujet d'étude actif.

L'étude des réseaux d'automates est movitée par leur application en biologie dans le contexte de l'étude de réseaux de régulation de gènes (Kauffman 1969 ; Thomas 1973 ; Demongeot, Goles, Morvan et al. 2010 ; Mendoza et Alvarez-Buylla 1998 ; Davidich et Bornholdt 2008). Dans ces réseaux, chaque automate représente un gène et la façon dont l'automate est mis à jour représente la façon dont les autres gènes l'influencent, que ce soit par activation ou par inhibition. Lorsque l'on considère l'interaction d'un ensemble de ces gènes, le comportement du système devient difficile à prédire : cela vient du nombre exponentiel de configurations possibles du système par rapport à sa taille. Le domaine des réseaux d'automates cherche donc principalement des méthodes pour permettre la prédiction de la dynamique d'un réseau sans la nécessité d'en calculer l'entièreté.

Ainsi, la littérature des réseaux d'automates s'est concentrée sur des structures de réseaux précises (Noual 2012 ; Alcolei, Perrot et Sené 2016), et a permis la prédiction de certains aspects de leur dynamique (Aracena 2008 ; Demongeot, Noual et Sené 2012 ; Aracena, Richard et Salinas 2017), ainsi que la description en complexité de questions associées (N. Alon 1985 ; Floreen et Orponen 1989 ; Orponen 1992 ; Bridoux, Durbec, Perrot et al. 2019 ; Bridoux, Gaze-Maillot, Perrot et al. p. d. ; Ilango, Loff et Oliveira 2020 ; Noûs, Perrot, Sené et al. 2020). Ces travaux de complexité permettent la mesure de la difficulté de la prédiction du comportement limite des réseaux étudiés, par exemple la prédiction de l'existence de points fixes dans leur dynamique.

Dans ce chapitre, nous développons les définitions associées aux réseaux d'automates, à leur mise à jour et à l'étude de leur dynamique.

4.2 Ensembles d'automates, états et configurations

Les *réseaux d'automates* sont définis sur un ensemble d'*automates*. Par convention, nous appellons cet ensemble S. Chaque élément dans S correspond à un automate dans le réseau. Par convention également, les éléments de S sont notés par des lettres minuscules, commençant par a. Bien que, dans le cadre de ce mémoire, les réseaux d'automates soient définis sur des ensembles finis d'automates, il est tout à fait concevable de définir S comme un ensemble infini dénombrable.

Pour un réseau d'automates donné, chaque automate dans S possède un *état*. Le champs des états possibles est défini par l'ensemble d'états de ce réseau d'automates. Cet ensemble est par convention noté Λ. Dans le cas où $\Lambda = \{0,1\}$, le réseau d'automates en question est un réseau d'automates booléens.

Le rassemblement des états de chaque automate du réseau constitue la *configuration* du réseau dans sa globalité. Cette configuration est formellement définie comme un vecteur sur S avec valeurs dans Λ. L'espace de ces vecteurs est noté Λ^S. Nous noterons en général une telle configuration x.

Définition 19 (Configuration)**.** *Soit S un ensemble d'automates et Λ un ensemble d'états. Une* configuration *est un vecteur défini sur Λ^S.*

4.3 Fonctions locales et graphe d'interaction

Étant donné un ensemble d'automates et un ensemble d'états, les *fonctions locales* définies par ces ensembles sont toutes les fonctions qui prennent en entrée la configuration du réseau, et qui retournent un état.

Définition 20 (Fonction locale)**.** *Soit S un ensemble d'automates et Λ un ensemble d'états. Une* fonction locale *est une fonction f définie sur $\Lambda^S \to \Lambda$.*

Enfin, un réseau d'automates définit une fonction locale à chaque automate s dans S.

Définition 21 (Réseau d'automates)**.** *Soit S un ensemble d'automates et Λ un ensemble d'états. Un* réseau d'automates *associe à tout $s \in S$ une fonction locale $f_s : \Lambda^S \to \Lambda$.*

Exemple 28. *Soit $S = \{a,b,c\}$ et $\Lambda = \{0,1\}$. Soit F le réseau d'automates qui à a, b et c associe les fonctions suivantes : $f_a(x) = \neg x_a$, $f_b(x) = x_a \vee x_c$, et $f_c(x) = \neg x_a \wedge \neg x_c$.*

Exemple 29. *Soit $S = \{d,e,g\}$ et $\Lambda = \{0,1\}$. Soit F' le réseau d'automates qui à d, e et g associe les fonctions suivantes : $f'_d(x) = \neg x_g$, $f'_e(x) = \neg x_d$, et $f'_g(x) = x_e$.*

Les réseaux d'automates sont des objets complexes aux interactions nombreuses. Afin d'aider notre compréhension de ces objets, nous utilisons une représentation graphique des réseaux d'automates sous forme de graphe. Dans ce graphe, nommé

graphe d'interaction, chaque automate est un sommet, et il y aura une arête orientée d'un sommet s vers un sommet r si et seulement si la valeur de s dans la configuration influence le calcul de la fonction locale de r. Si c'est le cas, on dit que s *influence* r, ce qui est défini comme suit.

Définition 22 (Influence)**.** *Soit F un réseau d'automates, et $s, r \in S$. On dit que s* influence *r si et seulement s'il existe une paire de configuration x et x' telle que $x_q = x'_q$ pour $q \neq s$ et $f_r(x) \neq f_r(x')$.*

Dans le cas booléen, l'influence peut être définie plus précisément dans le cas d'une influence positive d'un automate par un autre, ou d'une influence négative d'un automate par un autre. Ces notions émergent naturellement de l'application des réseaux d'automates dans l'étude de réseaux de régulation génétique, dans lesquels un gène va généralement permettre l'activation ou l'inhibition d'autres gènes.

Définition 23 (Influence positive)**.** *Soit F un réseau d'automates booléens, et $s, r \in S$ tels que s influence r. On dit que s* influence positivement *r si et seulement si pour toute paire de configuration x et x' telle que $x_q = x'_q$ pour $q \neq s$, on a*

$$x_s < x'_s \iff f_r(x) \leq f_r(x').$$

Définition 24 (Influence négative)**.** *Soit F un réseau d'automates booléens, et $s, r \in S$ tels que s influence r. On dit que s* influence négativement *r si et seulement si pour toute paire de configuration x et x' telle que $x_q = x'_q$ pour $q \neq s$, on a*

$$x_s < x'_s \iff f_r(x) \geq f_r(x').$$

Exemple 30. *Soit $S = \{a, b, c\}$ un ensemble d'automates. Rappellons le réseau d'automates F de l'exemple 28 qui à a, b et c associe les fonctions suivantes : $f_a(x) = \neg x_a$, $f_b(x) = x_a \vee x_c$, et $f_c(x) = \neg x_a \wedge \neg x_c$. Pour étudier l'influence de c sur les automates du réseau, détaillons l'ensemble des paires de configurations x, x' telles que $x_q = x'_q$ pour $q \neq c$:*

$$\{(000, 001), (010, 011), (100, 101), (110, 111)\}.$$

Si l'on remplace chacune de ces configurations x par $f_a(x)$, on obtient

$$\{(1, 1), (1, 1), (0, 0), (0, 0)\}.$$

Si on les remplace par $f_b(x)$, on obtient

$$\{(0, 1), (0, 1), (1, 1), (1, 1)\},$$

et si on les remplace par $f_c(x)$, on obtient

$$\{(1, 0), (1, 0), (0, 0), (0, 0)\}.$$

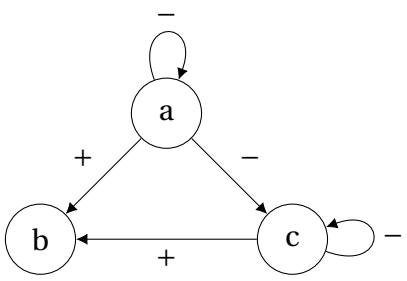

FIGURE 4.1 – Graphe d'interaction du réseau d'automates développé dans l'exemple 28.

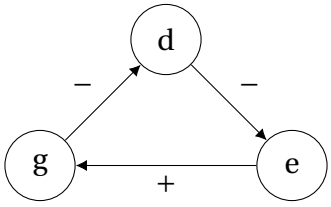

FIGURE 4.2 – Graphe d'interaction du réseau d'automates développé dans l'exemple 29.

On peut voir que c influence b et c, mais pas a. En effet, simplement modifier la valeur de x_c n'a aucun effet sur le calcul de $f_a(x)$. On peut également voir que l'influence de c sur b est positive, et que l'influence de c sur lui même est négative.

Il est possible qu'un automate s influence un automate r, sans pour autant que cette influence soit positive ou négative. Une telle influence est dite *non-monotone*.

Ces relations d'influence nous permettent de décrire le graphe d'interaction de tout réseau. Cette définition nous donne une représentation intuitive de la propagation de l'information à travers le réseau, et ce même graphe sera utilisé comme outil de preuve à maintes reprises.

Définition 25 (Graphe d'interaction)**.** *Soit F un réseau d'automates. Le* graphe d'interaction *de F est le graphe orienté qui est défini sur les sommets S, et qui possède une arête de s vers r si et seulement si s influence r. Si s influence positivement r, alors l'arête (s,r) est étiquetée par "+". Si s influence négativement r, alors l'arête (s,r) est étiquetée par "-".*

Les exemples 28 et 29 sont représentés sous la forme de graphes d'interaction dans les figures 4.1 et 4.2 respectivement.

Lorsque l'on peut représenter les interactions d'un réseau d'automates booléens par le biais d'arêtes positives et négatives, il devient intéressant de considerer le signe des cycles qui constituent le réseau. Le signe d'un cycle est simplement le produit des signes des arêtes qui le composent. L'étude de tels graphes d'interaction sans

nécessairement disposer des réseaux d'automates qui les définissent est un sujet de recherche actif (PAULEVÉ et RICHARD 2012 ; GADOULEAU et RICHARD 2016 ; ARACENA, RICHARD et SALINAS 2017 ; RICHARD 2018).

Définition 26 (Cycle signé)**.** *Soient F un réseau d'automates booléens et G son graphe d'interaction tels que les arêtes de G sont signées. Soit c un cycle de G et k le nombre d'arêtes au signe négatif qui le composent. Le cycle c est dit* positif *si k est pair, et* négatif *sinon.*

Exemple 31. *Considérons le réseau d'automates F' développé dans l'exemple 29, et son graphe d'interaction représenté en figure 4.2. Le cycle (d, e, g) dispose de deux arêtes négatives, (d, e) et (g, d). Ce nombre étant pair, le cycle (d, e, g) est donc positif.*

4.4 Mises à jour et exécutions

Un réseau d'automates est par nature un objet dynamique. Étant donné une configuration, la complexité du modèle apparaît lorsque l'on étudie les mises à jour successives du réseau sur cette configuration. Il n'existe cependant pas qu'une seule façon de mettre à jour un réseau ; en effet, de par sa nature décentralisée, chaque automate dans le réseau peut être considéré comme un acteur indépendant des autres, et l'on pourrait vouloir mettre à jour certains de ces acteurs sans mettre à jour les autres. Un automate seul est mis à jour en employant sa fonction locale ; comment exprimer la mise à jour du réseau dans sa totalité ?

Le protocole que l'on suit pour savoir quel automate mettre à jour à quel moment est appellé un *mode de mise à jour*. Dans le cadre de ce mémoire, nous définissons ces modes comme des séquences de mise à jour successives. Une mise à jour est un sous-ensemble des automates du réseau, de façon à ce que lorsqu'un automate s est inclus dans une mise à jour, cela signifie que cet automate exécutera sa fonction locale lors de cette mise à jour.

Définition 27 (Mise à jour)**.** *Soit F un réseau d'automates. Une* mise à jour *de F est un sous-ensemble $\delta \subseteq S$.*

Ainsi, mettre à jour un réseau d'automates d'après une mise à jour $\delta = \{a\}$ signifira l'application de la fonction locale f_a seule. La mise à jour $\delta = S$ signifira l'application de toutes les fonctions locales simultanément. L'application d'une mise à jour est une notion intuitive qui est définie comme suit.

Définition 28 (Application de mise à jour)**.** *Soit F un réseau d'automates, x une configuration et δ une mise à jour. L'*application de la mise à jour δ *sur la configuration x dans F est une nouvelle configuration notée $F_\delta(x)$, qui est définie par :*

$$\forall s \in S, F_\delta(x)_s = \begin{cases} f_s(x) & si\ s \in \delta \\ x_s & sinon \end{cases} .$$

L'application de la mise à jour $F_\delta(x)$ pour $\delta = S$ est également appellée la mise à jour *parallèle* du réseau d'automates F. Par confort nous la noterons $F(x)$.

Exemple 32. *Considérons le réseau d'automates F développé dans l'exemple 28, ainsi que trois mises à jour $\delta_1 = \{a\}$, $\delta_2 = \{b, c\}$ et S. Nous observons les faits suivants :*

$$\begin{aligned} F_{\delta_1}(000) &= 100 \\ F_{\delta_2}(000) &= 001 \\ F_S(000) &= 101 \\ F_{\delta_1}(101) &= 001 \\ F_{\delta_2}(101) &= 110 \\ F_S(101) &= 010 \end{aligned}$$

Il est naturel de vouloir mettre à jour un réseau d'automates plus d'une fois à la suite. Nous appellons une séquence de mises à jour un *mode de mise à jour*.

Définition 29 (Mode de mise à jour)**.** *Un* mode de mise à jour *est une séquence $\Delta = (\delta_1, \delta_2, \ldots)$, où pour tout k, δ_k est une mise à jour.*

Un tel mode de mise à jour peut être fini ou infini. Ces modes de mise à jour peuvent être classifiés selon certaines propriétés.

Définition 30 (Mode de mise à jour équitable)**.** *Soit Δ un mode de mise à jour. On appelle Δ un mode de mise à jour* équitable *si, dans le cas où Δ est fini, tout automate est inclus dans au moins une mise à jour de Δ, et si Δ est infini, chaque automate est inclus dans un nombre infini de mises à jour de Δ.*

Définition 31 (Mode de mise à jour parallèle)**.** *Le mode de mise à jour* parallèle*, noté Δ_P, est défini par $\Delta_P = (\delta)$ avec $\delta = S$.*

Le mode de mise à jour parallèle est alternativement appellé le mode de mise à jour *synchrone*.

Définition 32 (Mode de mise à jour séquentiel)**.** *Un mode de mise à jour Δ est dit* séquentiel *si et seulement si les mises à jour qui le composent sont des singletons et si leur ensemble est une partition de S.*

Définition 33 (Mode de mise à jour bloc-séquentiel)**.** *Un mode de mise à jour Δ est dit* bloc-séquentiel *si et seulement si l'ensemble des mises à jour qui le composent est une partition de S.*

On notera que, par définition, le mode de mise à jour parallèle et tout mode de mise à jour séquentiel sont également des modes de mise à jour blocs-séquentiels.

Exemple 33. *Soit $S = \{a, b, c\}$. Le mode de mise à jour parallèle est défini par $\Delta_P = (\{a, b, c\})$. Les modes de mise à jour $(\{a\}, \{b\}, \{c\})$, $(\{c\}, \{b\}, \{a\})$ et $(\{b\}, \{a\}, \{c\})$ sont des exemples de modes de mise à jour séquentiels, et $(\{b, c\}, \{a\})$ est un exemple de mode de mise à jour bloc-séquentiel.*

L'application d'un mode de mise à jour sur un réseau d'automates en partant d'une configuration donnée est appellée une *exécution* de ce réseau d'automates. Cette exécution est définie comme l'application successive des mises à jour qui composent le mode de mise à jour.

Définition 34 (Exécution)**.** *Soit F un réseau d'automates, $\Delta = (\delta_1, \delta_2, \ldots)$ un mode de mise à jour fini et x une configuration. On appelle exécution de F selon Δ sur x la configuration définie par les équations récursives suivantes :*

$$\begin{cases} F_{(\,)}(x) & = x \\ F_{(\delta_1, \delta_2, \ldots)}(x) & = F_{(\delta_2, \ldots)}(F_{\delta_1}(x)) \end{cases}.$$

Exemple 34. *Nous considérons F le réseau d'automates développé dans l'exemple 28. Soit $\Delta_1 = (\{a\}, \{b\}, \{c\})$ et $\Delta_2 = (\{b, c\}, \{a\})$. Nous observons les faits suivants :*

$$\begin{aligned} F_{\Delta_P \cdot \Delta_P \cdot \Delta_P}(000) &= 101 \\ F_{\Delta_1}(000) &= 110 \\ F_{\Delta_2}(000) &= 101 \\ F_{\Delta_P \cdot \Delta_P \cdot \Delta_P}(111) &= 010 \\ F_{\Delta_1}(111) &= 010 \\ F_{\Delta_2}(111) &= 010 \end{aligned}$$

L'exemple 34 illustre avec précision un fait connu dans la littérature des réseaux d'automates : le choix du mode de mise à jour fait varier drastiquement le calcul opéré par le réseau, et ce même en restreignant notre étude aux modes de mise à jour blocs-séquentiels voire même en comparant différent modes de mise à jour séquentiels (Robert 1986 ; Goles et Salinas 2008 ; Demongeot, Noual et Sené 2012 ; Aracena, Gómez et Salinas 2013 ; Noual et Sené 2018).

4.5 Graphe de la dynamique et attracteurs

Lorsque nous disposons d'un réseau d'automates et d'un certain mode de mise à jour, il est intéressant de représenter l'intégralité du calcul opéré par le réseau, en mettant en relation toutes les possibles configurations dudit réseau. Étant donné un mode de mise à jour Δ, le graphe de la dynamique d'un réseau d'automates est

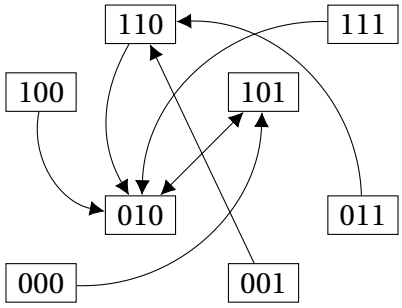

FIGURE 4.3 – Graphe de la dynamique parallèle du réseau d'automates développé dans l'exemple 28.

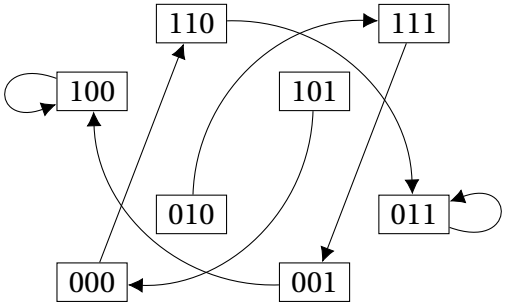

FIGURE 4.4 – Graphe de la dynamique parallèle du réseau d'automates développé dans l'exemple 29.

défini comme le graphe orienté dont les sommets sont les configurations du réseau et, pour deux configurations, le graphe disposera d'une arête orientée si et seulement si la première configuration, mise à jour sur le réseau avec Δ, donne la seconde configuration.

Définition 35 (Graphe de la dynamique)**.** *Soit F un réseau d'automates et Δ un mode de mise à jour. Le* graphe de la dynamique *de F selon Δ est le graphe orienté avec sommets dans Λ^S, tel que toute paire de configuration (x, y) est une arête du graphe si et seulement si $F_\Delta(x) = y$.*

La figure 4.3 représente le graphe de la dynamique selon le mode de mise à jour parallèle du réseau d'automates développé dans l'exemple 28.

La dynamique de modes de mise à jour fixés n'est pas le seul moyen d'étudier la dynamique d'un réseau d'automates. Il est en effet concevable de mettre à jour un réseau d'automates non pas par simple répétition d'un mode de mise à jour fini, mais par l'application non déterministe des fonctions locales des différent nœuds. Cette dynamique bien plus complexe peut également être représentée par ce qui est appellé un *graphe de la dynamique asynchrone*. Ce graphe possède une arête entre deux configurations étiquetée par un sommet s si et seulement si l'application de la

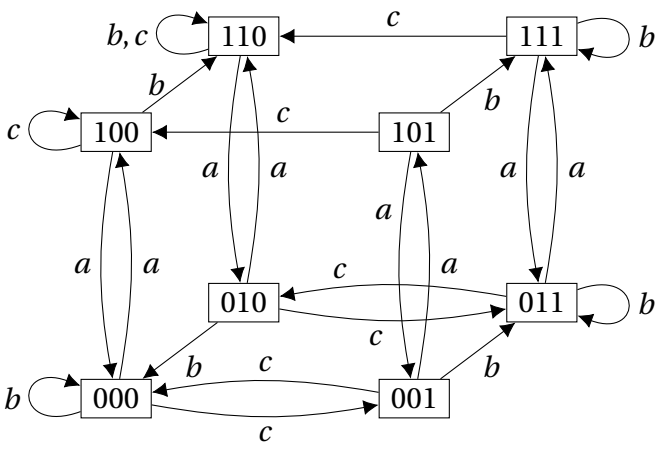

FIGURE 4.5 – Graphe de la dynamique asynchrone du réseau d'automates développé dans l'exemple 28.

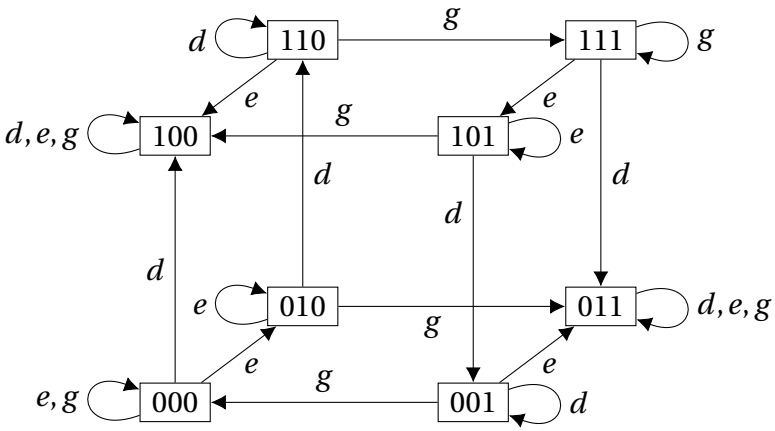

FIGURE 4.6 – Graphe de la dynamique asynchrone du réseau d'automates développé dans l'exemple 29.

fonction locale f_s seule fait passer de la première configuration à la seconde.

Définition 36 (Graphe de la dynamique asynchrone)**.** *Soit F un réseau d'automates. Le* graphe de la dynamique asynchrone *de F est le graphe orienté étiqueté avec sommets dans Λ^S et étiquettes dans S tel que (x, a, y) est une arête étiquetée du graphe si et seulement si $F_{\{a\}}(x) = y$.*

Les figures 4.5 et 4.6 représentent les graphes des dynamiques asynchrones des réseaux d'automates développés dans les exemples 28 et 29 respectivement.

Le graphe de la dynamique asynchrone représente toutes les façons de mettre à jour le réseau en utilisant des mises à jours successives qui ne concernent qu'un seul

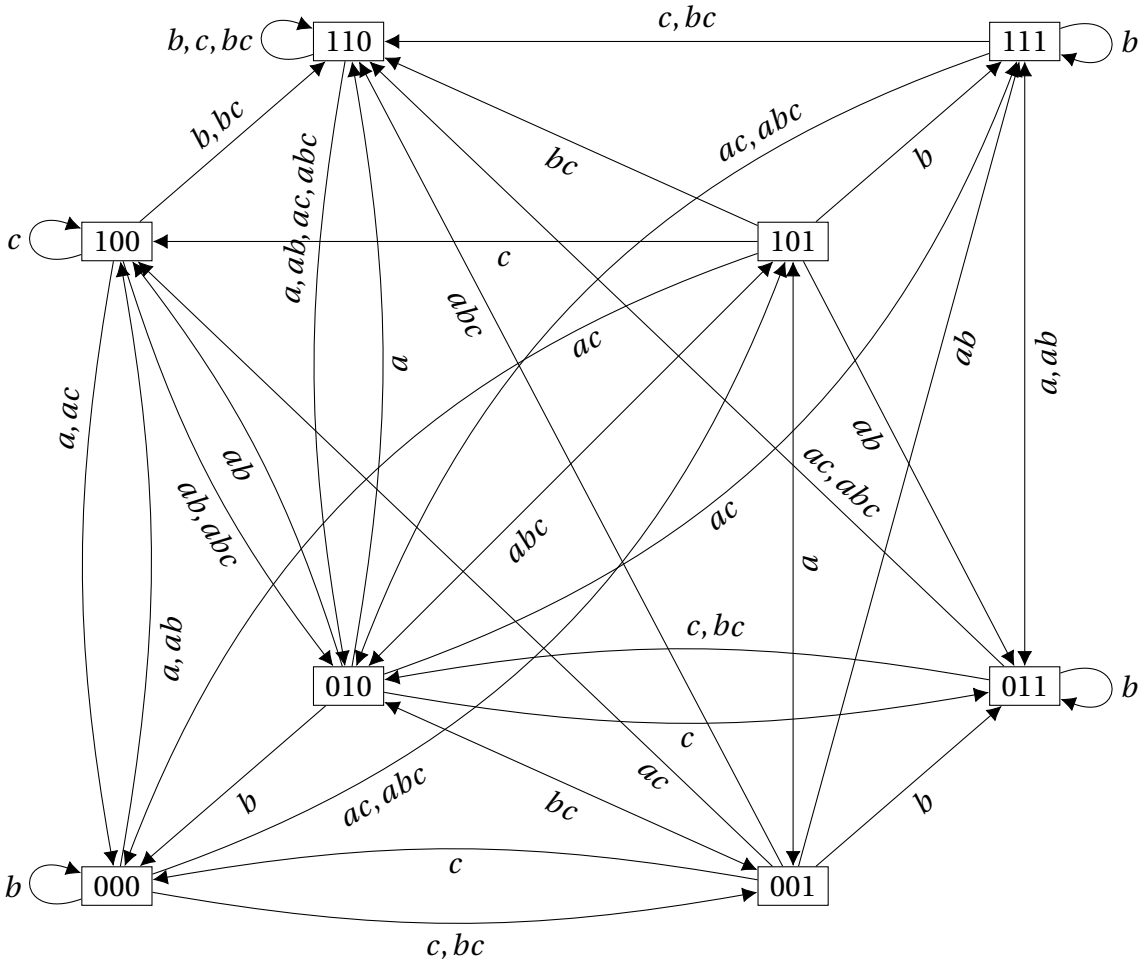

FIGURE 4.7 – Graphe de la dynamique totale du réseau d'automates développé dans l'exemple 28. L'étiquette ab représente la mise à jour $\delta = \{a, b\}$.

automate. Ainsi, ce graphe contient implicitement la dynamique de tous les modes de mise à jour sequentiels, entre autres.

Il est possible de représenter l'ensemble total de tous les modes de mises à jour sous la forme d'un graphe de la dynamique. Pour cela, il suffit d'étendre les étiquettes du graphe de la dynamique asynchrone à n'importe quel sous-ensemble de S. Ainsi, l'effet de n'importe quelle mise à jour depuis n'importe quelle configuration est répertorié. Nous appellons un tel graphe le *graphe de la dynamique totale*.

Définition 37 (Graphe de la dynamique totale)**.** *Soit F un réseau d'automates. Le* graphe de la dynamique totale *de F est le graphe orienté étiqueté avec sommets dans Λ^S et étiquettes dans $\wp(S) \setminus \emptyset$ tel que (x, δ, y) est une arête étiquetée du graphe si et seulement si $F_\delta(x) = y$.*

La figure 4.7 représente le graphe de la dynamique totale du réseau d'automates développé dans l'exemple 28.

Lorsque nous développons les détails de la dynamique d'un réseau d'automates, la complexité et la densité de l'information obtenue augmente. Ainsi, pour un réseau d'automates booléens de taille n, si les graphes de la dynamique obtenus sont tous des graphes de 2^n sommets, le nombre d'étiquettes contenues dans le graphe varie entre 2^n, $n \times 2^n$ ou 2^{2n}, en fonction de si l'on considère la dynamique d'un mode de mise à jour fixé, la dynamique asynchrone ou la dynamique totale respectivement. Dans les applications de ces modèles, le développement complet de ces dynamiques est ainsi très coûteux et presque toujours déraisonnable. Dans le cas non booléen à k états, ces mêmes dimensions deviennent k^n, $n \times k^n$ et k^{2n}.

Cependant, dans la plupart des applications des réseaux d'automates booléens, une compréhension extensive de la dynamique d'un réseau n'est pas nécessaire. En effet, on s'intéresse le plus souvent aux ensembles de configurations que l'on nomme les *attracteurs*. Ces configurations font parties de cycles de tailles variées ou de boucles dans le graphe de la dynamique du réseau. Ainsi, par exemple, dans le graphe représenté en figure 4.3, les configurations 010 et 101 font toute deux parties du même cycle de taille deux. Un tel cycle est appelé un *cycle limite*. Lorsqu'un attracteur est de taille 1, on l'appelle un *point fixe*.

Définition 38 (Attracteur)**.** *Soit G un graphe de dynamique. Un* attracteur *de G est une composante fortement connexe terminale de G.*

Définition 39 (Cycle limite)**.** *Soit F un réseau d'automates et Δ un mode de mise à jour. Un* cycle limite *est une séquence de configurations distinctes $(x_1, x_2, \ldots, x_k)$ telle que $F_\Delta(x_i) = x_{i+1}$ pour tout $0 < i < k$, $F_\Delta(x_k) = x_1$ et $k > 1$.*

Exemple 35. *Considérons le réseau d'automates F développé dans l'exemple 28. Le graphe de la dynamique parallèle de F, illustré en figure 4.3, dispose d'un seul attracteur, un cycle limite de taille 2 composé des configurations 010 et 101.*

Définition 40 (Point fixe)**.** *Soit F un réseau d'automates et Δ un mode de mise à jour. Un* point fixe *est une configuration x telle que $F_\Delta(x) = x$.*

Exemple 36. *Considérons le réseau d'automates F' développé dans l'exemple 29. Le graphe de la dynamique parallèle de F', illustré en figure 4.4, dispose de deux attracteurs qui sont des points fixes, les configurations 100 et 011.*

Dans le cas plus général des graphes de dynamique asynchrone ou totale, il est rare d'observer des cycles limites aussi clairement définis. On préfère donc en général définir le comportement limite d'un réseau par le biais d'attracteur complexe. Un attracteur complexe est un ensemble de configurations dont il est impossible pour le réseau de sortir.

Définition 41 (Attracteur complexe)**.** *Soit G un graphe de dynamique non-déterministe. Un* attracteur complexe *de G est une composante fortement connexe terminale de G de cardinal supérieur à 1.*

Un attracteur complexe est donc tout attracteur qui n'est pas un point fixe. Cette définition en invite une autre, celle du bassin d'attraction d'un attracteur, qui est composé de toutes les configurations en dehors de l'attracteur qui convergent asymptotiquement dans cet attracteur. Cette définition s'applique aux attracteurs complexes comme aux points fixes et aux cycles limites.

Définition 42 (Bassin d'attraction)**.** *Soit $G = (S, A)$ un graphe de dynamique, et S' un attracteur de G. Le bassin d'attraction de S' est l'ensemble des configurations $B_{S'}$ tel que :*

- *Pour tout $x \in B_{S'}$, et x' tel que $(x, x') \in A$, alors $x' \in B_{S'}$ ou $x' \in S'$.*
- *Pour tout $x \in B_{S'}$, il existe $x' \in S'$ et un chemin de x vers x' dans G.*

Exemple 37. *Considérons le réseau d'automates F' développé dans l'exemple 29. Le graphe de la dynamique asynchrone de F', illustré en figure 4.6, dispose de deux attracteurs qui sont des points fixes, les configurations* 100 *et* 011*. Le bassin d'attraction de ces deux attracteurs est l'ensemble des configurations* $\{000, 001, 010, 101, 110, 111\}$.

4.6 Résultats fondamentaux

Pour illustrer la difficulté des questions qui lui sont associées, nous proposons une présentation des résultats les plus fondateurs qui constituent notre compréhension actuelle de la dynamique des réseaux d'automates.

Il existe des cas particuliers de réseaux dont les attracteurs sont particulièrement bien compris. C'est le cas des réseaux d'automates acycliques, définis comme les réseaux d'automates dont le graphe d'interaction est acyclique. Ces objets sont étudiés en plus de détail au chapitre 6. Leur structure simple a permis la caractérisation suivante.

Théorème 1 (Robert 1980)**.** *Soit F un réseau d'automates. Si le graphe d'interaction de F est acyclique, alors sa dynamique totale n'a qu'un seul attracteur qui est un point fixe.*

Ce théorème démontre que les cycles dans le graphe d'interaction d'un réseau sont une condition nécessaire à la complexité de sa dynamique. Si un réseau est acyclique, alors toute configuration de ce réseau sous tout mode de mise à jour équitable converge vers un unique point fixe.

Les points fixes sont un sujet d'étude approfondi et s'avèrent particulièrement résilients au changement de mode de mise à jour. Le résultat suivant énonce que les points fixes d'un réseau d'automates sont les mêmes sous tout mode de mise à jour bloc-séquentiel. Cela se comprend intuitivement par le fait que, dans un point fixe,

aucun automate ne peut changer d'état, et ce, peu importe si ces automates sont mis à jour séparément ou en bloc.

Théorème 2 (ROBERT 1986)**.** *L'ensemble des points fixes d'un réseau d'automate est le même sous tout mode de mise à jour bloc-séquentiel.*

Ce théorème peut être étendu car un point fixe d'un réseau d'automates sous un mode de mise à jour bloc-séquentiel reste un point fixe sous toute itération du réseau, c'est à dire sous l'application de toute mise à jour.

Théorème 3 (Folklore)**.** *Soit F un réseau d'automate. Pour toute mise à jour δ, l'ensemble des points fixes de F sous un mode de mise à jour bloc-séquentiel est inclus dans l'ensemble des points fixes de F sous δ. La réciproque n'est pas vraie.*

La réciproque de ce théorème n'est pas vraie car certaines itérations peuvent provoquer l'apparition de points fixes par l'absence de la mise à jour d'un automate donné ou encore la répétition de la mise à jour d'un ou plusieurs automates au sein d'une période de mises à jour. Par exemple, le cas trivial de la mise à jour vide $\delta = \emptyset$ admet toute configuration comme point fixe, ce qui n'est évidemment pas le cas pour tout réseau d'automates mis à jour bloc-séquentiellement.

Ces premiers théorèmes sont simples et montrent que les points fixes de la dynamique d'un réseau d'automates sont résilients à la variation du mode de mise à jour. Reste cependant à comprendre les conditions spécifiques qui permettent l'observation d'un point fixe ou d'un attracteur complexe. Comme les cycles sont observés comme nécessaires à la complexité de la dynamique, il est pertinent de s'intéresser aux effets sur la dynamiques de la présence d'un cycle positif ou d'un cycle négatif dans le graphe d'interaction du réseau. Cette approche est représentée par les conjectures de Thomas, qui sont exprimées dans le contexte d'un mode de mise à jour asynchrone :

Conjecture 2 (THOMAS 1981)**.** *La présence d'un cycle positif dans le graphe d'interaction G d'un réseau F est nécessaire pour que F admette au moins 2 points fixes.*

Conjecture 3 (THOMAS 1981)**.** *La présence d'un cycle négatif dans le graphe d'interaction G d'un réseau F est nécessaire pour que F admette un attracteur complexe.*

Ces conjectures ont été prouvées dans le cadre booléen (REMY, MOSSÉ, CHAOUIYA et al. 2003), puis dans le cadre multivalué (RICHARD et COMET 2007 ; RICHARD 2010). La première conjecture, qui touche à la présence d'un cycle positif, s'avère vraie pour tout mode de mise à jour (NOUAL 2012 ; SENÉ 2012). En revanche, la conjecture qui touche à la présence d'un cycle négatif ne se généralise pas de cette façon car, par exemple, un circuit positif (un réseau d'automates composé seulement d'un cycle positif) admet des cycles limites sous le mode de mise à jour parallèle.

Ces conjectures et les théorèmes qui en découlent permettent d'exprimer deux intuitions : d'abord, les cycles positifs dans le graphe d'interaction d'un réseau sont nécessaires pour que ce réseau puisse se stabiliser de multiples façons ; les cycles positifs, en un sens, permettent une boucle de rétroaction positive qui font d'un réseau

un objet complexe mais stable. Ensuite, les cycles négatifs dans le graphe d'interaction d'un réseau sont nécessaires à l'apparition de comportements complexes instables. Ces résultats permettent également de souligner le rapport complexe entre mode de mise à jour et dynamique. Si les résultats qui portent sur les points fixes peuvent, par application des théorèmes 2 et 3, être généralisés à de nombreux modes de mise à jour, la caractérisation des cycles limites échappent à cette application.

La variation de la dynamique d'un réseau d'automates en fonction du mode de mise à jour étudié est un sujet actif du domaine. Par exemple, (GOLES et SALINAS 2008) établissent certaines différences significatives entre le mode de mise à jour parallèle et les modes de mise à jour séquentiels, en particulier dans leur application sur des réseaux structurés en couches. Similairement, (ARACENA, GOLES, MOREIRA et al. 2009) étudient certains critères qui permettent l'équivalence ou la différenciation de différents modes de mises à jours, et (NOUAL et SENÉ 2018) établissent un résultat qui touche à la sensibilité de la dynamique au synchronisme. Ces résultats s'appliquent à des classes spécifiques de réseaux d'automates. L'étude de l'effet du mode de mise à jour sur la dynamique est approfondi par (ARACENA, GOLES, MOREIRA et al. 2009 ; ARACENA, GÓMEZ et SALINAS 2013) qui proposent le formalisme de graphes de mise à jour, qui décrit le rapport entre le mode de mise à jour et le réseau d'automates, et permet de décrire des classes d'équivalence de modes de mises à jour. Certains problèmes de complexités reliés à ces graphes de mise à jour ont été caractérisés (ARACENA, FANCHON, MONTALVA et al. 2011 ; NOÛS, PERROT, SENÉ et al. 2020), tels que le problème de vérifier qu'un graphe donné est un graphe de mise à jour valide (qui est NP-complet), ou le problème de compter l'ensemble des étiquetages valide d'un graphe de mise à jour (qui est #P-complet).

Les travaux qui touchent au rapport entre variation du mode de mise à jour et variation de la dynamique expriment, dans leur ensemble, des solutions et des résultats qui s'appliquent à des cas particuliers et souvent spécifiques aux études concernées. Cela est attendu lors de l'étude d'une question aussi difficile, où l'absence de réponse générale et limpide fait converger la recherche vers des solutions plus simples dans des cas particuliers. Cependant, les travaux tels que (ARACENA, GÓMEZ et SALINAS 2013) nous montrent qu'il est encore possible de trouver des outils (ici les graphes de mise à jour) permettant d'aller plus loin dans la compréhension de l'influence de l'ordonnancement des mises à jour dans le temps.

Enfin, une portion de la littérature des réseaux d'automates pose la question du comptage de leurs attracteurs. Ces travaux établissent des bornes supérieures (ARACENA 2008 ; RICHARD 2009) ou comptent directement les attracteurs de classes particulières de réseaux d'automates (DEMONGEOT, NOUAL et SENÉ 2012 ; ARACENA, RICHARD et SALINAS 2017). Bien que l'approche de ces travaux constitue l'expression et la résolution partielle d'un simple problème de comptage, il peut être dit que l'expression la plus générale de ce problème de comptage – déterminer le nombre maximal d'attracteurs admis par un réseau d'automates – ouvre la voie vers une version discrète du second volet du seixième problème de Hilbert (ILYASHENKO 2002). La question générale du comptage des attracteurs de tout réseau d'automates s'avère une

question d'une redoutable difficulté, malgré sa résolution dans des cas de structures simples comme les réseaux d'automates acycliques, ou les circuits (c'est-à-dire les réseaux uniquement composés d'un cycle).