Simulation contexte et généralisations

La simulation est un concept singulier dans le calcul naturel. Bien que son intuition, la capacité de la reproduction du calcul, semble universelle, les définitions dont nous disposons sont toujours propres aux modèles qui les concernent. Ce schisme vient certainement du fait que nous n'avons jamais de définition unifiée de ce qu'est un calcul ; nous sommes réduits dans la pratique à ne définir que des exemples locaux de son apparition. En est-il de même avec la simulation, avec qui le calcul semble hautement apparenté ?

Nous ne sommes pas les premiers à nous poser la question (BOAS 2014). Il est désirable d'obtenir une telle définition générale si elle existe, simplement car l'unification d'une partie des méthodes d'un large domaine de recherche est à la clé. Devant une question aussi ambitieuse, nous adoptons la démarche de décrire les intuitions qui semblent être clés dans les définitions de simulation dont nous disposons dans certains modèles de calcul comme les automates cellulaires, puis de poursuivre la généralisation de ces intuitions vers, nous l'espérons, une définition applicable dans un cadre plus général.

2.1 Une définition informelle

La notion de simulation est fortement liée à la notion de l'universalité, dont il existe des variantes. La plus classique d'entre elles, l'universalité Turing, prend racine dans la thèse de Church-Turing, des deux mathématiciens qui, dans les années 1930, déclarent indépendamment avoir défini des modèles de calcul qui ont la faculté de reproduire tout calcul qu'un humain peut opérer si on lui procure une quantité infinie de feuilles, de crayon et de temps. Bien que cette thèse n'ait jamais été prouvée (car trop informelle), elle est communément admise et n'a jamais été contredite depuis 90 ans. Les machines de Turing sont parmi les exemples les plus connus de modèles dit Turing universels.

Une fois admise l'universalité Turing d'un modèle, il est possible d'étendre cette universalité à des modèles parfois de natures complètement différentes. Pour ce faire, on utilise la notion de simulation. La simulation est une relation entre modèles de calcul qui s'applique entre un modèle simulateur et un modèle simulé. En pratique, tout calcul opérable par la machine simulée peut être reconstruit comme un calcul opérable par la machine simulatrice, de façon à ce que le calcul de la machine simu-

latrice permette de prédire tout aspect du calcul simulé. Une telle relation se devra d'être réflexive et transitive : une machine doit être capable de reproduire son propre calcul, ce qui est trivial ; si une machine en simule une seconde qui en simule une troisième, alors elle simule la troisième. Ces deux propriétés font de la simulation un pré-ordre.

Une autre variante de l'universalité est nommée universalité intrinsèque, et ne fait sens que lorsque l'on examine la capacité d'un modèle à simuler tout autre modèle inclus dans la même famille. Cela est généralement dit d'un modèle spécifiquement capable d'une grande souplesse calculatoire. C'est le cas de certaines machines de Turing universelles, qui sont conçues pour accepter et reproduire le code décrivant n'importe quelle autre machine : une telle machine est intrinsèquement universelle dans la famille des machines de Turing. L'universalité intrinsèque ne découle cependant pas toujours de l'universalité Turing. Pour être intrinsèquement universel, un modèle devra souvent opérer son calcul en exploitant les propriétés intrinsèques de la famille de modèles utilisés. Les conditions spécifiques de ces propriétés changent avec la famille considérée, mais en général impliquent une simulation plus efficace.

Afin de se construire une intuition du fonctionnement de la simulation, examinons des définitions tirées de la littérature.

2.1.1 Simulation d'automates cellulaires

Les automates cellulaires sont une famille de modèles centrale dans le calcul naturel. Une présentation sommaire d'une fraction emblématique de ces modèles, les automates cellulaires élémentaires, est faite en section 1.7.

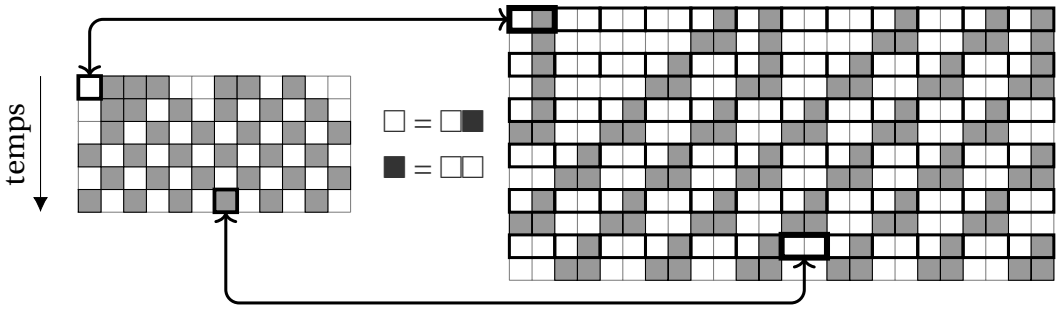

FIGURE 2.1 – Simulation d'une exécution de la règle 184 (gauche) par la règle 134 (droite). L'exécution de la règle 184 est encodée dans l'exécution de la règle 134. En ne considérant qu'une configuration sur deux, chaque paire de cellule (encadrées en noir sur la droite) encode une cellule de l'automate 184 de la façon suivante : l'état □□ encode la valeur ■ et l'état □■ encode la valeur □.

Lorsque l'on opère une simulation d'un automate cellulaire par un autre automate cellulaire, la méthode employée repose généralement sur l'encodage des valeurs des

cellules de l'automate simulé par des regroupements de cellules dans l'automate simulateur. La figure 2.1 détaille l'exemple de la simulation de la règle 184 par la règle 134 : ici, chaque automate de la règle 184 est remplacé par 2 cellules dans l'automate 134. On note également qu'une mise à jour supplémentaire est nécessaire entre chaque étape de la simulation. En d'autres termes, cette simulation opère par la mise à l'échelle dans l'espace et le temps de la règle 134. En choisissant une configuration initiale donnée par le bon encodage, la règle 134 se trouve reproduire "tous les calculs" de la règle 184.

Beaucoup d'autres simulations ont été développées dans la littérature des automates cellulaires, ce qui a permis l'identification de nombreux modèles intrinsèquement universels ou Turing universels (OLLINGER 2012 ; BECKER, MALDONADO, OLLINGER et al. 2018 ; COOK 2004 ; MARTIEL 2015). Un grand travail a été placé dans la généralisation des notions de simulation et d'universalité chez les automates cellulaires (OLLINGER 2002 ; THEYSSIER 2005), avec par exemple la généralisation de l'ensemble des méthodes reposant sur le groupage des cellules dans le simulateur par la notion de Bulking (DELORME, MAZOYER, OLLINGER et al. 2011a ; DELORME, MAZOYER, OLLINGER et al. 2011b), qui emploie des outils algébriques pour généraliser le plus fondamentalement possible la notion de mise à l'échelle de l'espace et du temps. Cette notion très générale de regroupement est ensuite utilisée pour définir plusieurs notions de simulation entre automates cellulaires.

La notion de Bulking permet une définition très générale de la simulation intrinsèque entre automates cellulaires ; c'est-à-dire toute simulation qui exploite les propriétés de localité et d'uniformité propres à ces modèles : leur nature uniforme et régulière invite à décrire une simulation efficace d'un automate par un autre comme une remise à l'échelle.

Cette définition est considérée satisfaisante pour englober les notions de simulation intrinsèque qui concernent les automates cellulaires. Qu'en est-il dans un cadre plus général ? Il existe par exemple des généralisations des automates cellulaires qui brisent la régularité de leur réseau ou l'uniformité de leur mise à jour dans l'espace ou le temps, comme les réseaux d'automates. Dans ces modèles, qui ne sont pas uniformes dans l'espace et le temps, le Bulking n'est pas applicable. De fait, l'intuition à la base de la notion de la simulation intrinsèque change radicalement dès que l'on change la famille de modèles que l'on considère.

2.1.2 Simulation de réseaux d'automates

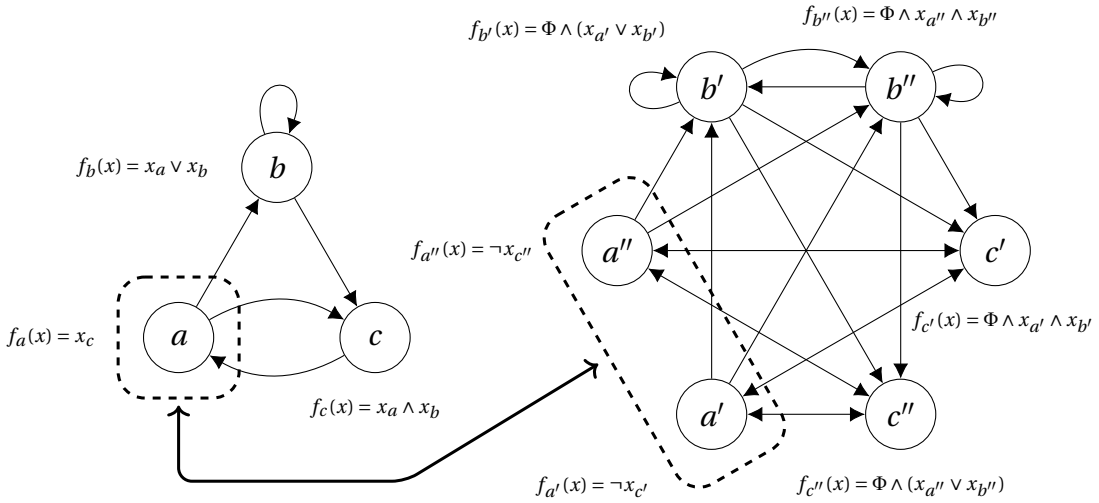

FIGURE 2.2 – Illustration de la simulation d'un réseau d'automates booléens de taille 3 (gauche) par un réseau d'automates booléens de taille 6 localement monotone (droite), dans lequel $\Phi = (x_{a'} \vee x_{a''}) \wedge (x_{b'} \vee x_{b''})$. Cette simulation est une application du corollaire 3 donné en section 5.3.3. L'état de chaque automate à gauche est encodé par les états de deux automates à droite ; par exemple, l'état de a est encodé par les états de a' et a''.

Les réseaux d'automates sont une variation des automates cellulaires qui ne sont pas définis sur un espace régulier. Si un automate cellulaire est composé de cellules, un réseau d'automates est composé d'éléments appelés automates. Les automates d'un réseau d'automates ne sont pas nécessairement mis à jour de façon homogène, ni même de façon synchrone. Un développement approfondi de ces modèles sera proposé au chapitre 4.

Puisque ces modèles ne garantissent pas la régularité de leur réseau ni celle de leur mise à jour, la réalisation d'une remise à l'échelle ne sera pas suffisante. À vrai dire, leur définition fait de chaque réseau d'automates un objet au graphe d'interaction particulier. Cependant, la plupart des simulations entre réseaux d'automates suit une logique similaire aux simulations entre automates cellulaires : chaque automate du réseau simulé sera représenté par un ou plusieurs automates dans le réseau simulateur. L'état de ce groupe d'automates encode alors l'état de l'automate simulé. Une illustration d'une telle simulation est représentée en figure 2.7, dans laquelle un réseau de taille 3 est simulé par un réseau de taille 6, tel que le comportement de chaque automate du réseau simulé est reproduit par une paire d'automates dans le simulateur. Cette méthodologie semble très similaire à celle développée dans l'exemple de la figure 2.1 : dans les deux cas, chaque automate est simulé localement par une petite structure. Une généralisation limitée de cette notion a été proposée par (PERROT, PERROTIN et

SENÉ 2018) et est également exprimée en section 5.3.3.

La simulation de tout réseau d'automates booléens par un réseau d'automates booléens localement monotone (PERROT, PERROTIN et SENÉ 2018) est un résultat qui permet l'établissement de l'universalité intrinsèque de ces derniers. La simulation de réseaux exécutés en parallèle par des réseaux exécutés par blocs-séquentiels (BRIDOUX, GUILLON, PERROT et al. 2017) en est un autre exemple. Une étude détaillée de la simulation dans les réseaux d'automates est faite par les travaux (BRIDOUX, CASTILLO-RAMIREZ et GADOULEAU 2020 ; BRIDOUX 2019 ; BRIDOUX, GADOULEAU et THEYSSIER 2020). Les notions liées aux réseaux d'automates sont développées en détail dans le chapitre 4.

2.2 Simulation par forme

La simulation intrinsèque semble en pratique souvent reposer sur une remise à l'échelle des modèles considérés : un élément simple est alors simulé par un ensemble d'éléments qui, mis ensemble, forment une structure au comportement recherché. Guidés par cette observation, nous développons une façon de percevoir les modèles de calcul qui permet la définition d'une notion de simulation, nommée simulation par forme. Son approche est très simple : les modèles de calcul sont perçus comme des collections de diagrammes espace-temps. Chacun de ces diagrammes assigne un état à chaque élément de l'espace et du temps. La simulation par forme repose sur l'encodage d'un tel diagramme par l'assemblage d'un diagramme simulateur obtenu par le remplacement de chaque élément atomique du diagramme simulé par une "forme".

2.2.1 Modèles de forme et sous-espaces fonctionnels

Comme décrit plus haut, les modèles de calcul sur lesquels la simulation par forme se définit sont considérés comme des collections d'exécutions. Dans cette section, l'ensemble X sera en général associé avec l'ensemble des positions dans l'espace et le temps d'un modèle de calcul. Dans beaucoup d'exemples, X est le produit cartésien d'un ensemble spatial P et d'un ensemble temporel T.

Exemple 17. *Considérons un automate cellulaire élémentaire. L'ensemble d'espace-temps X qui correspond à ce modèle est le produit cartésien $X = P \times T$, où $P = \mathbb{Z}$ correspond à l'espace d'une configuration infinie, et $T = \mathbb{N}$ correspond au temps, qui est infini avec une configuration initiale* 0. *Chaque paire $(p, t) \in X$ correspond au moment t de la cellule à la position p.*

L'ensemble S sera quant à lui associé à l'ensemble des états que chaque position dans l'espace-temps peut admettre.

Exemple 18. *Considérons un modèle d'automate cellulaire élémentaire. Ce modèle est booléen, et son espace d'états est $S = \mathbb{B}$.*

Dans le formalisme de la simulation par forme, une *exécution* d'un modèle est décrite par une fonction $e : X \to S$. Ici l'exécution e associe à tout élément dans l'espace-temps un état dans S. Un *modèle de forme* est défini par trois éléments : l'ensemble X, l'ensemble S, et l'ensemble des exécutions qui sont admises par le modèle. Ce dernier ensemble est généralement noté E, et est par définition un sous-ensemble de l'espace des fonctions $X \to S$. Nous appellons parfois E un *sous-espace fonctionnel*. Le modèle est décrit par le triplet $M = (X, S, E)$.

Exemple 19. *Soit $X = \mathbb{Z} \times \mathbb{N}$ et $S = \mathbb{B}$. Soit $f_k : \mathbb{B}^3 \to \mathbb{B}$ la fonction qui définit l'automate cellulaire élémentaire numéro k (voir définition en section 1.7). Le modèle de forme de cet automate est défini par le sous-espace fonctionnel E_k qui contient exactement les exécutions e telles que, pour tout x dans X et t dans T,*

$$e(x, t+1) = f_k(e(x-1, t), e(x, t), e(x+1, t)).$$

On note $M_k = (X, S, E_k)$ le modèle correspondant.

La définition d'un modèle de forme sélectionne les exécutions qui lui correspondent.

2.2.2 Formes

Les formes sont toutes les possibles macro-structures qui peuvent être construites dans l'exécution d'un modèle. Dans des exemples concrets de simulation, le comportement de ces macro-structures aura un comportement équivalent, dans le bon contexte, aux éléments atomiques du modèle simulé. Dans le cadre de la simulation par forme, nous proposons la définition de ces formes comme étant n'importe quelle disposition fixée d'éléments dans le modèle considéré. En ces termes, une forme est un ensemble de paires dans $X \times S$.

Définition 1 (Forme)**.** *Soit $M = (X, S, E)$ un modèle. Une* forme *h de M est un sous-ensemble de $X \times S$.*

Exemple 20. *Soit M_{134} le modèle d'automate cellulaire élémentaire défini dans l'exemple 19. Pour tout $b \in \mathbb{B}$, $x \in \mathbb{Z}$ et $t \in \mathbb{N}$, on définit la forme $h_{x,t,b}$ telle que*

$$h_{x,t,0} = \{((2x, 2t), 0), ((2x+1, 2t), 1)\}$$
$$h_{x,t,1} = \{((2x, 2t), 0), ((2x+1, 2t), 0)\}$$

Une forme peut être définie comme n'importe lequel de ces sous-ensembles. Ainsi, on pourra observer une forme vide, ou bien une forme qui définit deux valeurs à une seule position de l'espace et du temps. On pourra même considérer l'exemple extrême d'une forme qui définit toutes les paires possibles. Toutes ces choses sont des formes. Ainsi, l'ensemble des formes de M est simplement l'ensemble $\wp(X \times S)$.

Une forme est dite observée dans une exécution si et seulement si l'ensemble des paires (x, s) qui la composent se retrouvent dans l'exécution en question.

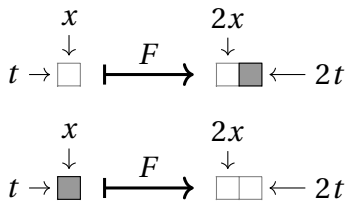

FIGURE 2.3 – Illustration de l'encodage opéré par la fonction de forme F définie dans l'exemple 22. Ici, les cases blanches représentent la valeur 0 et les cases noires représentent la valeur 1, similairement à la façon dont nous représentons les exécutions d'un automate cellulaire élémentaire.

Définition 2 (Observation de forme)**.** *Soit e une exécution et h une forme. La forme h est* observée *dans e lorsque $h \subseteq e$.*

Exemple 21. *On considère M_{134} le modèle défini dans l'exemple 19 et $h_{x,t,b}$ la famille de formes définie dans l'exemple 20. Pour toute exécution e de M_{134}, On remarque que pour tout $x \in \mathbb{Z}$, $t \in \mathbb{N}$, et tout triplet $(b_1, b_2, b_3) \in \mathbb{B}^3$,*

$$h_{x-1,t,b_1} \cup h_{x,t,b_2} \cup h_{x+1,t,b_3} \subseteq e \implies h_{x,t+1,f_{184}(b_1,b_2,b_3)} \subseteq e.$$

2.2.3 Fonction de forme

Pour décrire une simulation par un modèle $M = (X, S, E)$ d'un modèle $M' = (X', S', E')$, nous devons décrire la traduction des éléments atomiques de M' en termes de formes de M. Nous utilisons pour cela une fonction de forme.

Définition 3 (Fonction de forme)**.** *Une* fonction de forme F *de M' dans M est une fonction définie avec domaine $(X' \times S')$ et co-domaine $\wp(X \times S)$.*

Exemple 22. *Soit M_{134} notre modèle simulateur et M_{184} notre modèle simulé, définis dans l'exemple 19. Soit $h_{t,x,b}$ l'ensemble de formes défini dans l'exemple 20. Nous définissons F la fonction de forme qui vérifie*

$$F((x,t),b) = h_{t,x,b}.$$

Le calcul de cette fonction est représenté dans la figure 2.3.

La fonction de forme est fondamentalement locale ; elle permet la traduction d'un seul élément atomique du modèle simulé dans le modèle simulateur. Cette fonction peut également se comprendre comme une fonction d'encodage. Nous utilisons la fonction de forme dans les deux sens ; si elle permet d'obtenir l'encodage d'un élément simple, nous l'utilisons également pour déduire d'une exécution du modèle simulateur l'ensemble des éléments atomiques encodés dans cette exécution. Ce procédé est nommé interprétation de forme.

2.2.4 Interprétation de forme

Si la fonction de forme F est une fonction locale d'encodage, l'interprétation de forme repose sur une fonction f_F qui identifie l'ensemble des éléments atomiques encodés dans une exécution donnée du simulateur. Cette fonction de décodage est définie comme suit pour h toute forme de M :

$$f_F(h) = \{(x', s') \mid F(x', s') \subseteq h\}.$$

En général, cette fonction est définie sur $f_F : \wp(X \times S) \to \wp(X' \times S')$, ce qui signifie qu'elle admet l'ensemble des formes du simulateur en domaine et l'ensemble des formes du simulé en co-domaine.

Intuitivement, f_F effectue une collection de l'ensemble des éléments atomiques (x', s') dont la forme $F(x', s')$ est observée dans h. Ainsi, lorsque l'on prend une exécution e de M et qu'on l'interprète comme un ensemble de paires du domaine et co-domaine de e, l'évaluation de $f_F(e)$ retourne une forme composée de l'union des paires qui produisent par F les formes observées dans e. Dans certains cas, la forme retournée par $f_F(e)$ est complète dans le sens qu'il existe exactement une valeur $s' \in S'$ telle que $(x', s') \in f_F(e)$, pour tout $x' \in X'$. Dans ce contexte, il est valide de considérer $f_F(e)$ comme une exécution e' de M', auquel cas e' est l'interprétation de e d'après F.

2.2.5 Simulation par forme

Pour qu'un modèle M simule un modèle M', le choix de la fonction de forme F doit respecter un certain nombre de propriétés. D'abord, toute exécution de e' doit avoir une exécution de e dont elle est l'interprétation d'après F. Ensuite, il ne doit pas exister d'exécution e telle que l'interprétation de e est une fonction totale, qui n'est pas une exécution. Ce dernier point est important pour que la simulation comporte précisément l'information contenue dans le modèle M.

Définition 4 (Simulation par forme)**.** *Le modèle M* simule *le modèle M', noté $M' \preccurlyeq M$, si et seulement s'il existe une fonction de forme F de M' dans M telle que*

1. *Pour toute e' dans E', il existe e dans E telle que $f_F(e) = e'$.*
2. *Pour toute e dans E, si $f_F(e) \in X' \to S'$ (c'est à dire si $f_F(e)$ est fonctionnelle et totale), alors $f_F(e) \in E'$.*

La première condition est simple : elle requiert que le décodage donné par F permette de retrouver l'ensemble des exécutions de M' depuis M. La seconde condition est plus complexe : elle stipule que l'interprétation de toute exécution de M doit résulter en une forme qui doit soit être une exécution de M', soit être incomplète ou contradictoire. Dans le cas où cette condition n'est pas vérifiée, c'est qu'il existe une exécution e telle que $f_F(e)$ est une fonction $e' : X' \to S'$ bien définie qui n'est pas une exécution admise par M', ce qui est problématique car cela démontre que la simulation de M' par M "rajoute" de l'information au calcul de M' sans en respecter

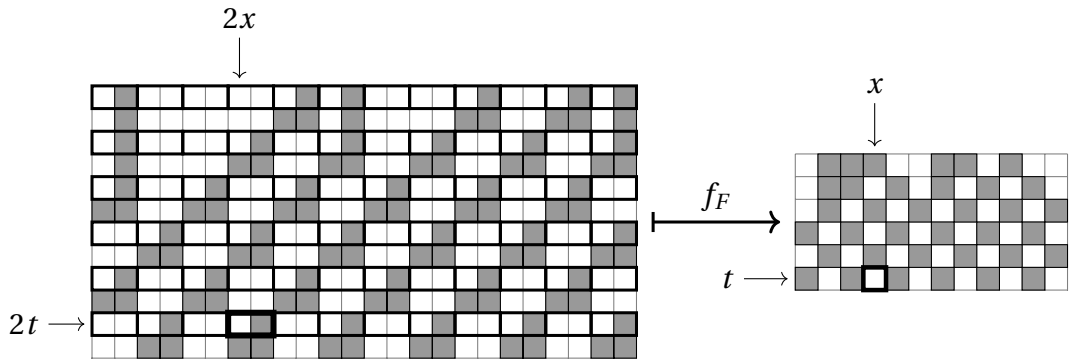

FIGURE 2.4 – Interprétation d'une exécution du modèle M_{134} (gauche) en une forme du modèle M_{184} (droite) par le biais de la fonction d'interprétation f_F dérivée de la fonction de forme F définie dans l'exemple 22. La forme obtenue dans cette illustration est un fragment valide d'une exécution du modèle M_{184}.

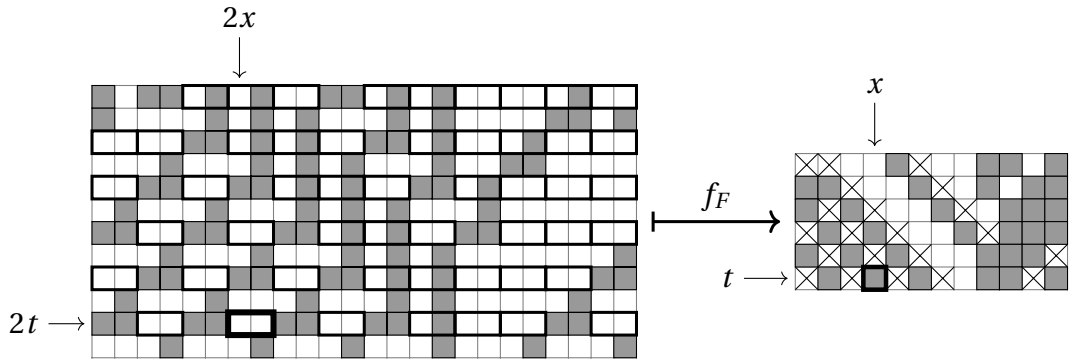

FIGURE 2.5 – Interprétation d'une exécution du modèle M_{134} (gauche) en une forme du modèle M_{184} (droite) par le biais de la fonction d'interprétation f_F dérivée de la fonction de forme F définie dans l'exemple 22. La forme obtenue dans cette illustration est incomplète : chaque croix représente un point dans l'espace et le temps qui n'a pas de valeur définie par f_F.

les règles. Cette seconde condition permet de vérifier que toute interprétation d'une exécution de M est soit une exécution admise par M', soit une forme incomplète ou contradictoire, et donc non considérée dans le contexte de la simulation.

Exemple 23. *Soient M_{134} et M_{184} les modèles de forme et F la fonction de forme définis dans l'exemple 22. Nous reportons l'observation de l'exemple 21 que pour tout $x \in \mathbb{Z}$, $t \in \mathbb{N}$, et tout triplet $(b_1, b_2, b_3) \in \mathbb{B}^3$,*

$$h_{x-1,t,b_1} \cup h_{x,t,b_2} \cup h_{x+1,t,b_3} \subseteq e \implies h_{x,t+1,f_{184}(b_1,b_2,b_3)} \subseteq e.$$

Par cette observation, nous établissons que $M_{184} \preccurlyeq M_{134}$. En effet, pour toute configuration initiale dans M_{184}, il existe une exécution de M_{134} dont la configuration intiale l'interprète par la fonction de forme F. L'application de l'observation permet de déduire que le reste de l'exécution de M_{184} est également proprement interprété.

On observe également qu'aucune exécution de M_{134} ne peut être interprétée de façon contradictoire; c'est-à-dire que les formes $F((x,t),0)$ et $F((x,t),1)$ ne peuvent jamais être définies dans la même exécution. Cela signifie que l'interprétation de toute exécution de M_{134} est soit incomplète, soit complète; dans le second cas, notre précédente observation implique qu'il s'agit d'une exécution valide de M_{184}, et la seconde condition de simulation est donc vérifiée.

Cet exemple est une application de la simulation par forme sur un exemple de simulation classique illustré en figure 2.1.

Une application de la fonction d'interprétation f_F dérivée de la fonction de forme F définie dans l'exemple 22 est illustrée en figures 2.4 et 2.5. La figure 2.4 représente une exécution du modèle M_{134} qui est interprétée par f_F en une exécution valide du modèle M_{184}. La figure 2.5 représente une exécution e du modèle M_{134} dont l'interprétation par f_F donne une information incomplète, ce qui signifie que l'exécution e ne simule aucune exécution du modèle M_{184}.

2.2.6 Critiques

Intuitivement, la simulation est comprise comme une relation qui démontre qu'un modèle est au moins aussi expressif qu'un autre : puisque tout modèle est au moins aussi expressif que lui-même, la réflexivité est une propriété qui tombe sous le sens. De façon similaire, la transitivité affirme que si un premier modèle est démontré aussi expressif qu'un second modèle, qui est démontré aussi expressif qu'un troisième modèle, alors le premier modèle est aussi expressif que le troisième modèle. Ces deux propriétés mises ensemble font de la simulation un pré-ordre. Sont-elles vérifiées en pratique dans le contexte de la simulation par forme ?

Propriété 1. *Tout modèle de forme M vérifie $M \preccurlyeq M$.*

Démonstration. La fonction de forme $F(x,s) = \{(x,s)\}$ permet pour tout modèle de forme de définir la fonction d'interprétation f_F suivante :

$$\begin{aligned} f_F(e) &= \{(x,s) | F(x,s) \subseteq e\} \\ &= \{(x,s) | \{(x,s)\} \subseteq e\} = e \end{aligned}$$

ce qui est l'identité sur les formes de M et qui vérifie trivialement toutes les conditions de la simulation par forme. □

Si la réflexivité est facile à vérifier, il n'en est pas de même pour la transitivité. Pour trois modèles M, M', M'' tels que $M' \preccurlyeq M$ et $M'' \preccurlyeq M'$, et pour F et F' les fonctions de forme de ces simulations respectives, une proposition naturelle de démonstration

que $M'' \preccurlyeq M$ repose sur la création d'une fonction de forme F'' définie comme une forme de composition de F et F' comme suit :

$$F''(x'', s'') = \{F(x', s') | (x', s') \in F'(x'', s'')\}.$$

Cette fonction applique simplement la fonction de forme F à tous les éléments de la forme retournée par la fonction de forme F'. Notons cette forme de composition la *composition naturelle* de F et F'.

Propriété 2. *Il existe M, M', M'' des modèles de forme et F, F' des fonctions de forme tels que F vérifie $M' \preccurlyeq M$ et F' vérifie $M'' \preccurlyeq M'$, mais tels que la composition naturelle de F et F' ne vérifie pas $M'' \preccurlyeq M$.*

Démonstration. Nous établissons un contre-exemple pour démontrer le résultat. On considère $M = M_{134}$, $M' = M_{184}$ et F tels que définis dans l'exemple 22. On construit $M'' = (X'', S'', E'')$ le modèle de forme qui vérifie $X'' = \{a, b\}$, $S'' = \{0, 1, 2, 3\}$ et $e'' \in E''$ si et seulement si $e''(a) \geq 2$ et $e''(a) \geq e''(b)$. Pour démontrer que $M'' \preccurlyeq M_{184}$, On construit F' la fonction de forme qui vérifie

$$\begin{aligned}
F'(a,2) &= \{((1,0),0)\} \\
F'(a,3) &= \{((1,0),1)\} \\
F'(b,0) &= \{((0,1),0),((1,1),0)\} \\
F'(b,1) &= \{((0,1),0),((1,1),1)\} \\
F'(b,2) &= \{((0,1),1),((1,1),0)\} \\
F'(b,3) &= \{((0,1),1),((1,1),1)\}.
\end{aligned}$$

Pour vérifier cette simulation, observons que, pour toute exécution e' de M_{184}, si $F'(a,2) \in e'$, alors la case 1 de la configuration initiale de e' est à 0. En théorie, si la valeur de a est 2, alors la valeur de b peut être tout sauf 3. Cela est vérifié en pratique puisque l'observation de $F'(a,2) \in e'$ implique par la règle locale de l'automate cellulaire élémentaire 184 que les cases 0 et 1 de la seconde configuration de e' peuvent admettre toutes valeurs sauf 1 et 1. On en déduit que si $F'(a,2) \in e'$, alors $F'(b,k) \in e'$ est possible si et seulement si $k \leq 2 = a$.

Lorsque l'on suppose $F'(a,3) \in e'$, les cases 0 et 1 de la configuration suivante ne sont pas restreintes, et b peut être observé avec n'importe quelle valeur de 0 à 3, ce qui vérifie $M'' \preccurlyeq M_{184}$.

On définit maintenant F'' comme la composition naturelle de F et F' :

$$F''(x'', s'') = \{F(x', s') | (x', s') \in F'(x'', s'')\}.$$

Soit e_I l'exécution de M_{134} qui possède pour configuration initiale la répétition du motif 1, 0, 0. Cette exécution, interprétée par f_F, donne une information incomplète et ne change donc pas la nature de la simulation $M_{184} \preccurlyeq M_{134}$. Cependant, son interprétation par $f_{F''}$ donne une forme de M'' qui est une fonction bien définie, vérifiant

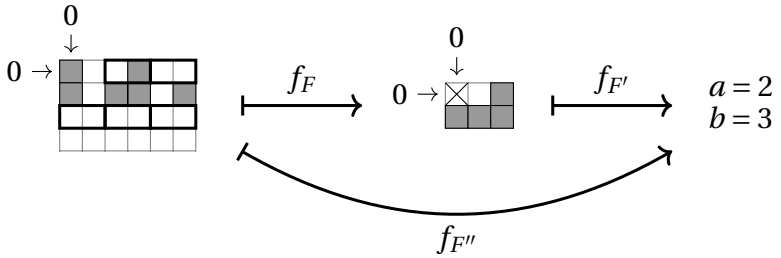

FIGURE 2.6 – Illustration du contre-exemple développé dans la preuve de la propriété 2, comprenant une exécution du modèle M_{134} (gauche), une forme du modèle M_{184} (centre) et une forme du modèle M'' (droite). La croix dans la forme du centre indique que l'exécution de gauche est interprétée par f_F de façon incomplète. De plus, l'interprétation de cette forme incomplète par $f_{F'}$ donne une exécution invalide de M'', avec $a < b$. On en déduit que l'interprétation de l'exécution de gauche par $f_{F''}$ donne une exécution invalide du modèle M'', ce qui implique que F'' ne définit pas une simulation.

$f_{F''}(a) = 2$ et $f_{F''}(b) = 3$, ce qui n'est pas une exécution valide de M'' (voir figure 2.6). La fonction de forme F'', définie comme la composition naturelle de F et F', ne permet donc pas de vérifier $M'' \preccurlyeq M_{134}$.

□

Si la composition de F et F' ne garantit pas une simulation, la question de la transitivité de la simulation par forme reste ouverte. En effet, il reste à décider l'existence d'une autre fonction de forme F'', qui ne serait pas la composition naturelle de F et de F', et qui permet une simulation. Il ne parait pas facile de trouver une méthode simple pour générer cette fonction F'' ; et s'il n'en existe aucune, la mise en pratique de la simulation par forme devient difficile.

Question ouverte 1. *La simulation par forme est-elle transitive ?*

D'une façon plus générale, ce formalisme tente de ne pas considérer l'axe du temps comme une dimension identifiée de ses modèles. Cette particularité rend la représentation de concepts comme la causalité, le déterminisme, ou même des notions plus triviales comme la configuration initiale ou finale d'un calcul impossible dans le cas le plus général.

De plus, ce formalisme se limite à des interprétations fondée sur la composition simple de formes prédéfinies dans l'espace et le temps. Cela peut avoir pour conséquence d'empêcher l'inclusion de certains résultats de simulation dès lors qu'ils concernent des modèles dans lesquels la position dans l'espace-temps de certaines macro-structures ne sont pas garanties avec une précision absolue. Parmi les modèles concernés, nous pouvons citer les modèles de particules, dans lesquels le calcul prend la forme d'interaction de particules mouvantes. Dans la plupart de ces modèles, les

détails précis du calcul considéré peuvent décaler une particule donnée selon la situation, sans entraver la simulation en cours (COOK 2004). Pourtant, la simulation par forme sera incapable d'interpréter cette particule de façon uniforme si sa position exacte n'est pas garantie.

2.3 Préordre de simulation

Parmi les nombreuses préoccupations du domaine de l'informatique théorique réside la motivation de généraliser et comprendre la notion de langage de programmation. Ce domaine large englobe de très nombreux modèles tels que les automates finis (SALOMAA 2014) ou le lambda-calcul (HINDLEY et SELDIN 2008). Lorsqu'il devient nécessaire de comparer le calcul de deux machines, ce domaine utilise la notion très générale de bisimulation ; qui considère que deux modèles sont bisimilaires si toute opération faite par l'un peut être reproduite par l'autre (MILNER 1971 ; SANGIORGI 2011).

Dans cette section, nous explorons une définition de bisimulation et proposons des modifications qui permettent de l'employer dans le contexte des modèles de calcul naturel.

Le travail présenté dans cette section est le fruit d'une recherche en collaboration avec Guilhem Gamard, Pierre Guillon, Kévin Perrot, Enrico Porreca, Martín Ríos Wilson, Sylvain Sené, Guillaume Theyssier et d'autres.

2.3.1 Définitions

La bisimulation, et la relation de préordre de simulation qui la compose, sont des concepts qui permettent la comparaison du calcul des systèmes de transition d'états. Un système de transition d'états est un graphe orienté, dont les sommets et les arêtes sont étiquetés.

Définition 5 (Système de transition d'états)**.** *Pour S un ensemble d'états et Λ un ensemble d'étiquettes, un* système de transition d'états *est un triplet $(S, \Lambda, \to)$, avec $\to$ un sous-ensemble de $S \times \Lambda \times S$.*

Les sommets de ce graphe sont parfois appelés états ou processus, et les étiquettes des arêtes parfois appelées actions. Un triplet $(s, a, s') \in \to$ s'interprète en disant que depuis l'état s, l'action a permet d'évoluer dans l'état s', ce que l'on note $s \xrightarrow{a} s'$. Lorsque Λ est un singleton, on considère que le système n'est pas étiqueté, et on considère alors $\to$ comme une simple relation $\to \subseteq S \times S$.

Exemple 24. *Soit S l'ensemble des configurations $\mathbb{B}^{\mathbb{Z}}$, Λ un singleton et k un entier entre 0 et 255. On définit $\to_{ECA(k)}$ la relation qui vérifie $s \to s'$ si et seulement si s' s'obtient par mise à jour de l'automate cellulaire élémentaire numéro k sur la configuration s. Le triplet $(S = \mathbb{B}^{\mathbb{Z}}, \Lambda = \{a\}, \to_{ECA(k)})$ est le système de transition d'états qui correspond à l'automate cellulaire élémentaire numéro k.*

La bisimulation est une relation qui se définit entre deux états d'un unique système. Ces états sont dits bisimilaires si toute action opérable dans un de ces états peut être opérée dans l'autre de façon à ce que les nouveaux états obtenus soient également bisimilaires.

Définition 6 (Bisimulation)**.** *Pour* $(S, \Lambda, \rightarrow)$ *un système de transition d'états, une* bisimulation *est une relation* $\mathscr{R} \subseteq S \times S$ *qui vérifie que, dès que* $s \mathscr{R} r$*, de façon symétrique*

— *pour tout* s' *tel que* $s \xrightarrow{a} s'$*, il existe* r' *tel que* $r \xrightarrow{a} r'$ *et* $s' \mathscr{R} r'$

— *pour tout* r' *tel que* $r \xrightarrow{a} r'$*, il existe* s' *tel que* $s \xrightarrow{a} s'$ *et* $s' \mathscr{R} r'$*.*

La *bisimilarité*, notée $\sim$, est l'union de toutes les bisimulations. C'est-à-dire que $s \sim r$ implique qu'il existe une bisimulation $\mathscr{R}$ telle que $s \mathscr{R} r$.

Une façon de lire ces définitions est que deux états s et s' font partie d'une bisimulation si et seulement si toute action de l'un peut être reproduite par une action de l'autre. Cette définition s'intéresse en un sens à l'équivalence de deux modèles dynamiques non pas par leur calcul final, mais par l'équivalence et l'inter-reproductibilité de chacune de leurs étapes de calcul.

La bisimulation est naturellement définie comme la juxtaposition de deux relations : que le calcul de s puisse être reproduit par s', et vice-versa. La version de la bisimulation qui casse cette symétrie est appellée préordre de simulation, et sa définition est intuitivement dérivée de celle de la bisimulation.

Définition 7 (Préordre de simulation)**.** *Pour* $(S, \Lambda, \rightarrow)$ *un système de transition d'états, un* préordre de simulation *est une relation* $\mathscr{S} \subseteq S \times S$ *telle que* $s \mathscr{S} r$ *implique que pour tout* s' *tel que* $s \xrightarrow{a} s'$*, il existe* r' *tel que* $r \xrightarrow{a} r'$ *et* $s' \mathscr{S} r'$*.*

Ce formalisme permet de définir que le calcul à partir d'un état peut être reproduit en tout point à partir d'un autre. Cette notion commence à s'approcher d'une notion de simulation dans le contexte des modèles de calcul naturel. Cependant, dans la plupart des simulations de la littérature des modèles de calcul naturel, le calcul des modèles considérés n'a pas de correspondance étape de calcul par étape de calcul. On peut prendre pour exemple la simulation représentée en figure 2.1, dans laquelle deux étapes de la règle 134 sont nécessaires pour reproduire une étape de la règle 184. Tenter d'appliquer la définition de préordre de simulation sur ces modèles vus comme des systèmes de transition d'états ne permettra pas de représenter la majorité des simulations qui ont été développées dans le domaine.

Nous nous intéressons donc à une variation de ce préordre de simulation qui permet l'inclusion de plus d'une étape de calcul pour reproduire l'étape opérée par le modèle simulé. Cette variation est le préordre de simulation faible, défini comme suit.

Définition 8 (Préordre de simulation faible)**.** *Pour* $(S, \Lambda, \rightarrow)$ *un système de transition d'états et* $\tau \in \Lambda$ *une action dite silencieuse, un* préordre de simulation faible *est une relation* $\mathscr{S} \subseteq S \times S$ *telle que* $s \mathscr{S} r$ *implique*

— *Pour tout* s' *tel que* $s \xrightarrow{\tau} s'$*, il existe* r' *tel que* $r \xrightarrow{\tau^*} r'$ *et* $s' \mathscr{S} r'$*.*

— *Pour tout s' tel que $s \xrightarrow{a} s'$, il existe r' tel que $r \xrightarrow{\tau^* a \tau^*} r'$ et $s' \mathscr{S} r'$.*

Dans cette définition, la notation τ^* correspond à l'application de l'étoile de Kleene sur l'action τ qui décrit la répétition de cette opération un nombre fini de fois. Dans le contexte du préordre de simulation faible, l'action dite silencieuse τ correspond à des actions intermédiaires qui ne sont pas considérées comme de véritables actions observables du modèle concerné. Ainsi on pourra considérer la concaténation d'un nombre arbitraire d'action silencieuses τ^* comme une opération effectivement silencieuse, ce qui permet en pratique de concaténer plusieurs opérations en une seule et permettre de rendre la définition de simulations entre modèles plus souple.

2.3.2 Travail d'adaptation

La bisimulation et ses définitions dérivées sont fondées sur des notions propres à des domaines assez différents du domaine du calcul naturel. Cela implique certaines différences formelles qu'il faut observer lorsque l'on souhaite proposer une adaptation de ces définitions d'un domaine vers l'autre.

Une différence notable est la définition des étiquettes des transitions. Dans le contexte des modèles de calcul, différents modèles risquent d'avoir des actions radicalement différentes ; on citera par exemple l'ensemble des modes de mise à jour possibles pour un réseau d'automates, ou l'ensemble des choix d'une machine de Turing non déterministe. Il n'y a pas de raison de trouver une nomenclature universelle pour tous ces modèles bien distincts, à raison que le choix de cette nomenclature a une influence sur les simulations qui pourraient être alors définies. Ainsi, pour tenter de produire une définition pratique et générale, nous introduisons une fonction g qui permet la traduction de l'action simulée en une série d'actions du simulateur. Cela permet également d'éviter la convention de l'action silencieuse, qui ne fait que rarement sens dans le contexte des modèles de calcul naturel.

Il est également intéressant de noter que les définitions présentées dans la sous-section précédente s'appliquent uniquement sur les états d'un seul système. Cela permet une perspective générale, car la simulation entre deux modèles peut être comprise comme la simulation de paires d'états dans le système de transition d'états construit comme l'union des systèmes de chaque modèle séparé.

Enfin, comme nous ne nous intéressons ici qu'à la simulation d'un modèle par un autre, il nous semble plus simple de remplacer la relation $\mathscr{S}$ par une fonction partielle que nous appellons h. Cette fonction a pour domaine un sous-ensemble des états du modèle simulateur, et pour co-domaine les états du modèle simulé.

Définition 9 (Simulation)**.** *Pour $(S, \Lambda, \rightarrow)$ un modèle A et $(S', \Lambda', \rightarrow')$ un modèle B, le modèle A est* simulé *par le modèle B s'il existe deux fonctions $g : \Lambda \rightarrow \Lambda'^*$ et $h : S' \twoheadrightarrow S$ telles que h est surjective, et*

1. *pour tout $s, r \in S$, $a \in \Lambda$ et $s' \in dom(h)$, si $h(s') = s$ et $s \xrightarrow{a} r$, alors il existe $r' \in dom(h)$ tel que $h(r') = r$ et $s' \xrightarrow{g(a)}{}' r'$ (tout calcul est simulé)*

2. *pour tout $s \in S$, $a \in \Lambda$, et $s', r' \in dom(h)$, si $h(s') = s$ et $s' \xrightarrow{g(a)}{}' r'$ alors $s \xrightarrow{a} h(r')$ (tout calcul simulé est un calcul).*

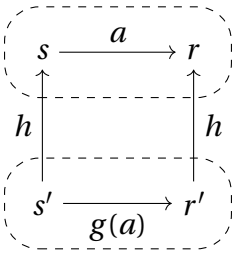

FIGURE 2.7 – Diagramme représentant les relations entre les états s, r d'un modèle simulé (en haut) et s', r' d'un modèle simulateur (en bas). La surjectivité de h implique que pour tout s, r, il existe s', r' tels que $h(s') = s$ et $h(r') = r$. La première clause de la définition 9 permet de vérifier que dès que l'on peut aller de s à r par l'action a, alors on peut aller de tout s' vers tout r' par la séquence d'actions $g(a)$. La seconde clause de la définition permet de vérifier que pour tout s', dès que l'on peut aller vers un r' par la séquence d'actions $g(s)$, alors on peut aller de s à r par l'action a.

Cette définition introduit deux fonctions, g et h, qui définissent la transformation qui permet la simulation d'un modèle par un autre. La fonction h est une fonction de décodage : pour s' une configuration du modèle simulateur, $h(s')$ calcule la configuration s du modèle simulé qui est encodée dans s'. Cette fonction est partielle, car toutes les configurations du simulateur n'ont pas vocation à encoder une configuration du modèle simulé. Cette fonction est surjective, car toute configuration simulée doit être encodée par au moins une configuration du simulateur. Elle n'est cependant pas nécessairement injective, car il peut exister plusieurs façons d'encoder une configuration donnée dans le simulateur. La fonction g, pour a toute action du simulé, calcule la séquence d'actions qui reproduit a dans le simulateur.

Nous proposons de créer des notions plus spécifiques de simulations en mesurant g et h en complexité.

Définition 10 (Simulation en espace logarithmique)**.** *On définit que A est simulé par B en espace logarithmique, noté $A \preccurlyeq_L B$, si A est simulé par B par le biais de fonctions g et h telles que*

- *g et h sont calculables en espace logarithmique*
- *pour tout état s de A, on peut calculer un état s' de B tel que $h(s') = s$ en espace logarithmique.*

Définition 11 (Simulation en temps polynomial)**.** *On définit que A est simulé par B en temps polynomial, noté $A \preccurlyeq_P B$, si A est simulé par B par le biais de fonctions g et h telles que*

— *g et h sont calculables en temps polynomial*
— *pour tout état s de A, on peut calculer un état s' de B tel que $h(s') = s$ en temps polynomial.*

Il est important de garantir que ces définitions restent des définitions de préordres, c'est-à-dire des relations binaires réflexives et transitives, ce que nous vérifions par les propriétés suivantes.

Propriété 3. *La simulation en espace logarithmique vérifie*
— *la réflexivité : tout modèle A vérifie $A \preccurlyeq_L A$;*
— *la transitivité : pour tous modèles A, B, C, si $A \preccurlyeq_L B$ et $B \preccurlyeq_L C$, alors $A \preccurlyeq_L C$.*

Démonstration. Soit A un modèle. En prenant h et g comme les identités sur S et Λ respectivement, nous vérifions que h est une bijection, et que $s \xrightarrow{a} r$ est équivalent à $h^{-1}(s) \xrightarrow{g(a)} h^{-1}(r)$. Ces fonctions, qui recopient simplement leur entrée sur leur sortie, sont bien calculables en espace logarithmique. Cela prouve que la simulation en espace logarithmique est réflexive.

Soient $A = (S, \Lambda, \rightarrow), B = (S', \Lambda', \rightarrow')$ et $C = (S'', \Lambda'', \rightarrow'')$ trois modèles. Par hypothèse, A est simulé en espace logarithmique par B par le biais des fonctions g et h, et B est simulé en espace logarithmique par C par le biais de fonctions g' et h'. Le reste de cette preuve consiste à prouver que nous pouvons simuler A par C en espace logarithmique par le biais de fonctions g'' et h''.

Nous construisons $h'' : S'' \nrightarrow S$ telle que $h''(s'') = (h \circ h')(s'')$, définie pour tout s'' qui vérifie $s'' \in \mathrm{dom}(h')$ et $h'(s'') \in \mathrm{dom}(h)$.

Nous construisons $g'' : \Lambda \rightarrow \Lambda''^*$ comme la composition naturelle de g et g', c'est à dire que pour tout $a \in \Lambda$, $g''(a)$ est la concaténation des séquences obtenues par l'application de g' sur chaque élément de la séquence $g(a)$, dans l'ordre.

Par hypothèse que h, h', g et g' sont calculables en espace logarithmique, les fonctions g'' et h'' sont également calculables en espace logarithmique.

Puisque par hypothèse pour tout s on peut construire s' en espace logarithmique tel que $h(s') = s$ et pour tout s' on peut construire s'' en espace logarithmique tel que $h'(s'') = s'$, on peut, pour tout s, construire s'' en espace logarithmique tel que $h''(s'') = s$. Enfin, h'' est surjective par construction.

Pour voir que ces fonctions définissent bien une simulation de A par C, considérons s, r dans S, a dans Λ et $s'' \in S''$ tels que $s \xrightarrow{a} r$, et $h''(s'') = s$.

Si $s \xrightarrow{a} r$, alors il existe $s', r' \in \mathrm{dom}(h)$ tels que $s' \xrightarrow{g(a)}{}' r'$ et $h(s') = s$ et $h(r') = r'$. Soit $s'_1, s'_2, \ldots s'_k$ une séquence d'états dans S' obtenue en démarrant à $s'_1 = s'$ et en terminant à $s'_k = r'$, en prenant la séquence d'actions $g(a)$. Par hypothèse, pour chaque paire (s'_i, s'_{i+1}) dans cette séquence, il existe une paire (s''_i, s''_{i+1}) dans les états du modèle C telle que $h'(s''_i) = s'_i$, $h'(s''_{i+1}) = s'_{i+1}$ et $s''_i \xrightarrow{g'(g(a)_i)}{}'' s''_{i+1}$. En étendant cette analyse à l'ensemble des paires successives dans la séquence $s'_1, s'_2, \ldots s'_k$, on en déduit qu'il existe $s''_i, r''_i \in \mathrm{dom}(h'')$, tels que $s''_i \xrightarrow{g''(a)}{}'' s''_k$, avec $h''(s''_i) = s$ et $h''(s''_k) = r$, ce qui implique que g'' et h'' vérifient la première clause de notre définition.

Maintenant, soit $s \in S$, $a \in \Lambda$, $s'' \in \mathrm{dom}(h'')$ et $r'' \in S''$. Supposons $h''(s'') = s$, et $s'' \xrightarrow{g''(a)}{}'' r''$. Pour démontrer que la seconde clause de notre définition s'applique, nous devons montrer que $r'' \in \mathrm{dom}(h)$ et $s \xrightarrow{a} h''(r'')$.

Pour ce faire, nous procédons à la même décomposition du chemin parcouru dans S'. Par hypothèse, et puisque $g''(a)$ se définit comme la concaténation des séquences obtenues par application de g' sur chaque élément de la séquence $g(a)$, décomposons cette concaténation. Soit $s'_1, \ldots, s'_k$ toute séquence de longueur $k = |g(a)|$ qui vérifie $h(s'_1) = s$, et pour tout $i < k$, $s'_i \xrightarrow{g(a)_i}{}' s'_{i+1}$. Par hypothèse, on sait que $s'_k \in \mathrm{dom}(h)$, et que $s \xrightarrow{a} h(s'_k)$. Soit $s''_1, s''_2, \ldots, s''_k$ la séquence d'états dans S'' qui pour tout $i < k$ vérifient $s''_i \xrightarrow{g'(g(a)_i)}{}'' s''_{i+1}$. On peut prouver par récurrence en démarrant par l'hypothèse $h'(s''_1) = s'_1$, et par hypothèse de la seconde clause de notre définition, que pour tout i, $s''_i \in \mathrm{dom}(h')$, $h'(s''_i) = s'_i$ et $s'_i \xrightarrow{g(a)_i}{}' h'(s''_{i+1})$. En mettant tout cela bout à bout, on obtient que $s''_k \in \mathrm{dom}(h')$, ce qui avec le fait que $h'(s''_k) = s'_k$ et que $s'_k \in \mathrm{dom}(h)$ implique que $s''_k \in \mathrm{dom}(h'')$, et que $s'_1 \xrightarrow{g(a)}{}' h'(s''_k)$, ce qui implique à son tour que $s \xrightarrow{a} h''(s''_k)$. Notre seconde clause est donc également vérifiée. □

Propriété 4. *La simulation en temps polynomial vérifie*
- *la réflexivité : tout modèle A vérifie $A \preccurlyeq_P A$;*
- *la transitivité : pour tous modèles A, B, C, si $A \preccurlyeq_P B$ et $B \preccurlyeq_P C$, alors $A \preccurlyeq_P C$.*

Démonstration. La preuve est analogue au cas logarithmique en observant que la composition de deux fonctions calculables en temps polynomial est également calculable en temps polynomial. □

La simulation linéaire englobe toute simulation dont la construction se limite à la mise à l'échelle du modèle simulé par une certaine constante. Cela inclut la simulation de la règle 184 par la règle 134. Il est cependant nécessaire d'observer que les systèmes de transition d'états définis dans l'exemple 24 qui correspondent aux automates cellulaires élémentaires ne peuvent vérifier aucune simulation mesurée en temps ; cela vient du fait qu'ils définissent des configurations infinies. Pour résoudre ce problème, nous définissons la variation suivante.

Exemple 25. *Pour n un entier, l'ensemble de configurations $\mathbb{B}^n$, l'ensemble d'actions $\{a\}$, et k un entier entre 0 et 255, on définit $\rightarrow_{ECA(n,k)}$ la relation qui vérifie $s \rightarrow s'$ si et seulement si s' s'obtient par mise à jour de l'automate cellulaire élémentaire numéro k sur la configuration finie s. Le triplet $R_{n,k} = (S = \mathbb{B}^n, \Lambda = \{a\}, \rightarrow_{ECA(k)})$ est le système de transition d'états qui correspond à l'automate cellulaire élémentaire numéro k sur des configurations finies périodiques de taille n.*

Pour rappel, l'exécution d'un automate cellulaire élémentaire sur une configuration finie est définie en section 1.7. Cette définition permet la vérification de la simulation illustrée en figure 2.1.

Exemple 26. *Prouvons $R_{n,184} \preccurlyeq_L R_{2n,134}$ pour tout entier n.*

Soit $h : \mathbb{B}^{2n} \twoheadrightarrow \mathbb{B}^n$ la fonction qui est définie sur toute configuration obtenue par la concaténation des mots 00 *et* 01*. La fonction h retourne alors une configuration de taille n obtenue par le remplacement des mots* 00 *par* 1 *et* 01 *par* 0*. On définit $g : \{a\} \to \{a\}^*$ la fonction qui vérifie $g(a) = aa$.*

Ces fonctions réalisent la simulation illustrée en figure 2.1, et sont calculables en espace logarithmique, puisque la fonction g est constante et que la fonction h est définie par une simple opération de réécriture. De plus, pour tout $s \in \mathbb{B}^n$, on peut construire $s' \in \mathbb{B}^{2n}$ tel que $h(s') = s$ en espace logarithmique simplement par l'application de la règle de réécriture $0 \mapsto 01, 1 \mapsto 00$.

Cette propriété est satisfaisante pour décrire cette simulation dans le contexte des systèmes de transition d'états.

2.4 Travail futur

Si la définition présentée dans ce chapitre nous semble prometteuse, nous mentionnons que les propriétés exposées dans ce chapitre ne sont pas suffisantes pour justifier la nature généralisatrice de cette définition. En effet, seul un travail d'application systématique de cette définition aux nombreux travaux du domaine du calcul naturel permettra de se convaincre de sa nature généralisatrice. Au cours de ce travail, et si l'on suppose que notre définition soit assez généralisante, il sera sans doute mis en lumière que les définitions de simulations exprimées pour les différents modèles de calcul naturel possèdent des variations et des subtilités qui préviennent une simple généralisation. Parmi ces subtilités, il est probable que la définition de simulation intrinsèque ne soit pas généralisable dans l'absolu. Bien que la caractérisation de la simulation intrinsèque comme une simulation particulièrement efficace soit intuitive, il reste à savoir si une limite précise de complexité permet cette caractérisation. Peut-être cette étude permettra-t-elle d'exhiber des exemples contre intuitifs d'une simulation acceptée comme intrinsèque mais ayant une plus grande mesure de complexité qu'une autre simulation non intrinsèque ?

Quelles que soient les réponses à ces questions, les définitions proposées dans ce chapitre sont des fragments incomplets qui demandent un travail approfondi pour en cerner les limites. Bien que nous n'ayons aucune garantie à ce stade que notre définition fonctionne dans ce cadre étendu, nous avons confiance que la direction générale de cette définition offre une direction pertinente. À la clé de cette recherche réside la création d'une hiérarchie formelle des modèles de calcul naturel qui permettra peut-être de donner une unité entre les domaines d'horizons très variés qui les régissent.