Cours n°2

1 Les stratégies de mise à jour des automates cellulaires

Au moins trois aspects sont à considérer (et donc décisions à prendre selon ce que l'on veut modéliser) :
— l'algorithme global de *mise à jour des cases*,
— le *déterminisme* ou indéterminisme des règles applicables
— et enfin l'application systématique ou seulement *probabilistes* des règles.

1.1 Mises à jour synchrones, asynchrones, bloc-synchrones...

Un automate cellulaire *synchrone* est un automate cellulaire dont on met à jour *toutes* les cases en même temps. *Toutes* les cases évoluent en même temps avec pour convention que la transformation d'une case ne dépend *que* de l'*ancien* état de ses cases voisines. Plusieurs phénomènes biologiques dont la dynamique est régie par des influences locales peuvent être simulés ainsi.

L'algorithme de mise à jour est simplement une boucle `for` sur tous les états. La structure de données est constituée de *deux copies* de l'espace considéré. Dans le cas particulier des coordonnées cartésiennes, on peut réaliser la boucle `for` globale *via* un enchevêtrement de n boucles `for`, où n est la dimension.

Asynchrones : on choisi une case après l'autre. Un ordre unique et déterministe de mise à jour ne serait pas très malin. Mieux vaut un choix aléatoire de la case à mettre à jour. On prend souvent une distribution uniforme mais ce n'est pas obligatoire (e.g. modéliser des « points chauds » de l'espace).

Bloc-synchrones : on découpe l'espace en « blocs » de cases on choisit à chaque étape un bloc à mettre à jour (de manière synchrone au sein du bloc). La succession des blocs traités est par contre asynchrone : on peut, là aussi, avoir un ordre unique ou des tirages aléatoires, uniformes ou non, proportionnels à la taille de chaque bloc, *etc.*

1.2 Règles déterministes ou non déterministes

Le jeu de la vie a donc ceci de simple que c'est un automate synchrone et déterministe : un état n'a qu'un seul futur possible. De tels automates cellulaires ne sont pas adaptés pour des simulations réalistes en biologie. Ne pouvoir écrire que des règles déterministes est très limitatif.

L'automate sera dit *non déterministe* s'il existe des états dans un voisinage où plusieurs règles peuvent s'appliquer. C'est un peu plus délicat à gérer mais souvent plus crédible biologiquement. Dans ce cas on fait, pour chaque case traitée, l'inventaire des règles applicables, ensuite on peut faire un tirage aléatoire (uniforme ou non) sur la règle à appliquer. Une solution courante est de mettre un poids sur chaque règle (généralement un nombre entier) et de faire un tirage à distribution proportionnelle à ces poids, *etc.*

On peut souvent (mais pas toujours, ça dépend de ce que l'on modélise) suivre le principe que s'il y a au moins une règle applicable alors on en appliquera une. C'est alors le choix de la règle à appliquer qui est probabiliste. Dans ce cas, on suit une syntaxe des règles comme : $\begin{array}{|c|c|c|}\hline x_1 & x_2 & x_3 \\ \hline x_4 & \text{c} & x_5 \\ \hline x_6 & x_7 & x_8 \\ \hline\end{array} \longrightarrow \boxed{\text{c'}}\ (P)$ où P est généralement un entier qui représente le *poids relatif* de la règle par rapport aux autres règles. Le tirage aléatoire de la règle à appliquer est alors fait selon une probabilité proportionnelle au poids de chaque règle applicable.

Si l'on veut s'affranchir des poids lorsque plusieurs règles peuvent s'appliquer dans certains états, on peut aussi avoir un système de priorité entre règles et appliquer systématiquement la règle la plus prioritaire. Dans ce cas, le choix de règle redevient déterministe.

Notons qu'un choix de règle déterministe ne rend pas pour autant l'automate sous-jacent déterministe. En effet pour qu'un automate soit déterministe, il faut qu'une seule transition à la fois soit possible dans chaque état : un choix non synchrone de la mise à jour des cases (section 1.1) ou une décision probabiliste d'application de règle (section 1.3 ci-dessous) peut rendre l'automate non déterministe même si le choix de règle à appliquer est unique.

1.3 Mises à jour probabilistes

Lorsqu'on a déterminé une règle à appliquer sur une case donnée selon les deux aspects précédents (synchronisme et choix de règle à appliquer). Il reste encore la possibilité de décider de l'appliquer ou non, avec une certaine probabilité d'application. C'est en particulier bien pratique pour modéliser des réactions biochimiques ayant des vitesses différentes : la probabilité d'application sera proportionnelle à la vitesse de réaction.
Par exemple, la naissance et la mort dans le jeu de la vie sont régis par des règles strictes et l'on préférerait clairement ajouter, entre autres, un peu d'aléatoire : avoir des probabilités de naissances et des probabilités de mort en fonction du voisinage.

On peut utiliser une forme de règles avec probabilité $\begin{array}{|c|c|c|}\hline x_1 & x_2 & x_3 \\ \hline x_4 & \text{c} & x_5 \\ \hline x_6 & x_7 & x_8 \\ \hline\end{array} \longrightarrow \boxed{\text{c'}}\ [p]$ où p est un nombre réel compris entre 0 et 1. Pour chaque règle, on se contente généralement d'une distribution uniforme de probabilité d'application de la règle dans tous l'espace (lorsqu'elle s'applique), cependant ou pourrait tout à fait imaginer des « compartiments » de l'espace avec des probabilités d'applications spécifiques.

Lorsque l'automate est déterministe, un algorithme de simulation apte à simuler cet automate probabiliste n'est pas très compliqué à réaliser : il suffit d'ajouter un tirage aléatoire qui peut éventuellement conduire à ne pas appliquer la modification induite par la règle.

Exemple : un automate cellulaire planaire où les états possibles des cases sont booléens, le voisinage réduit à Nord/Sud/Est/Ouest, et où une case prend pour état suivant celui de la majorité de ses voisines, avec une probabilité proportionnelle au nombre de ses voisines dans cet état. S'il y a deux voisines de chaque état, la case reste dans son état courant. Ainsi une case qui a trois voisines à `True` et une à `False` possède 75% de chance de passer à True et 25% de chances de rester à sa valeur courante.

Nota : pour optimiser la rapidité de simulation, on *normalise les probabilités* pour qu'une règle au moins possède une probabilité 1 (=100%). Il suffit de faire une règle de trois sur toutes les probabilités afin que celle de plus forte probabilité atteigne 100% : en pratique cela ne change pas notablement le comportement global mais rend la simulation plus rapide en temps de calcul.

1.4 . . . et autres stratégies originales . . .

Le cas général pour la syntaxe d'une règle, qui prend en compte tous les aspects cités précédemment, serait donc :

$$\begin{array}{|c|c|c|}\hline x_1 & x_2 & x_3 \\ \hline x_4 & \text{c} & x_5 \\ \hline x_6 & x_7 & x_8 \\ \hline\end{array} \longrightarrow \boxed{\text{c'}}\ (P)[p]$$

mais généralement il est trop compliqué de l'appliquer, sauf exemple particulièrement simple, comme celui de la section suivante.

La seule limite aux variantes possibles des automates cellulaires est l'imagination. . . Il existe des versions dans lesquelles la règle à appliquer change en fonction de la région dans l'espace où se trouve la case à mettre à jour. . .

Il faut cependant garder à l'esprit qu'il est presque impossible de prévoir quelle sera l'évolution d'un automate cellulaire, même dans les cas assez simples. En fait, il y reste encore quelques recherches actives pour étudier certaines propriétés des automates cellulaires 1D, synchrones non probabilistes. . . et le cas 2D est actuellement un domaine de recherche à part entière.

2 TD : Exemple d'automate cellulaire synchrone, non déterministe et probabiliste : modélisation de phénomènes de contagion

On se place dans le cadre des automates cellulaires synchrones, non déterministes, avec règles probabilistes. L'objectif est de modéliser les phénomènes de contagion dans une classe en supposant que les élèves ont une place attitrée qu'ils conservent d'un jour à l'autre.

Une case de l'automate cellulaire représentera donc une table+chaise individuelle de la classe ; la classe est supposée rectangulaire mais certaines places peuvent être inoccupées.

On souhaite traduire les informations suivantes :

1. Un élève peut tomber malade avec une probabilité égale à $(1+n)$ dixièmes, où $n = 0..8$ est le nombre de voisins contagieux dans la classe.
2. Lorsqu'un élève tombe malade, il est contagieux durant une journée exactement, puis il est absent 2 ou 3 jours (probabilités identiques entre 2 ou 3 jours), et revient « sain » mais avec une chance sur deux d'être encore contagieux (porteur sain) pendant une journée.
3. Après quoi, l'élève n'est non seulement plus malade ni contagieux, mais il est de plus immunisé.

Pour simplifier, on supposera qu'il n'y a ni week-end ni vacances.

Exercice 1 :
- Combien d'états différents faut-il prévoir pour chaque case ? donnez leur un nom explicite.
- En supposant que la classe contienne 50 places, combien l'automate associé a-t-il d'états possibles ?
- Quelle durée de temps représente une transition d'automate ?

Réponse : Innoccupé $-$, Sain S, Contagieux C, Absent_nbjours (nbjours=1..3) A_i, PorteurSain P, Immunisé I. Donc 8 états par case, soit 8^{50} états (plus de 10^{45} états possibles). Enfin 1 transition = 1 journée.

Exercice 2 : Décrivez l'ensemble des règles qui traduisent la première information. On s'autorisera toutes les conventions de notation et les itérations permettant de factoriser *sans ambiguïté* plusieurs règles.

Réponse :

$\neg(C \vee P)$	$\neg(C \vee P)$	$\neg(C \vee P)$
$\neg(C \vee P)$	S	$\neg(C \vee P)$
$\neg(C \vee P)$	$\neg(C \vee P)$	$\neg(C \vee P)$

$\longrightarrow \boxed{C}$ (1)[10%] *(NOTA : cette règle là est « déterministe » en ce sens qu'un voisinage qui a cette forme n'entre dans aucune des formes des règles qui suivent, donc c'est la seule règle qui peut s'appliquer et donc le poids est absolument arbitraire, pourvu que ce ne soit pas 0)*

$C \vee P$	$\neg(C \vee P)$	$\neg(C \vee P)$
$\neg(C \vee P)$	S	$\neg(C \vee P)$
$\neg(C \vee P)$	$\neg(C \vee P)$	$\neg(C \vee P)$

$\longrightarrow \boxed{C}$ (1)[20%] *(même remarque sur le poids arbitraire)*
et les 7 autres places possibles de $(C \vee P)$

$C \vee P$	$C \vee P$	$\neg(C \vee P)$
$\neg(C \vee P)$	S	$\neg(C \vee P)$
$\neg(C \vee P)$	$\neg(C \vee P)$	$\neg(C \vee P)$

$\longrightarrow \boxed{C}$ (1)[30%] *(et toujours même remarque sur le poids arbitraire)*
et toutes les autres combinaisons avec 2 $(C \vee P)$ parmi les 8 cases

etc. jusqu'aux 8 règles à 80% et la règle à 90% où toutes les cases voisines sont contagieuses.

Exercice 3 : Les règles précédentes n'ont pas de sens pour les cases au bord de l'automate. Trouvez une astuce pour éviter d'introduire des règles *ad hoc* pour ces cases.

Réponse : Il suffit d'entourer artificiellement avec des cases inoccupées $(-)$.

Exercice 4 : Où un élève a-t-il intérêt à se placer pour éviter la contagion autant que possible ?

Réponse : Un élève aux coins de la classe bénéficie des cases inoccupées qui diminuent sa probabilité d'être atteint par la contagion, à défaut les bords sont mieux que le centre. Du même coup, les « voisins des bords » ont plus de chance d'avoir des voisins sains, et ainsi de suite en allant vers le centre... et la contagion maximale est au centre.

Exercice 5 : Décrivez l'ensemble des règles qui traduisent les informations numérotées 2. et 3.

Réponse : C'est en fait facile une fois qu'on a bien choisi les états possibles de chaque case dans l'exercice 1... mais il faut cette fois faire appel à des règles non déterministes. Par exemple :

*	*	*
*	A_2	*
*	*	*

$\longrightarrow \boxed{A_3}$ (2)[100%]

*	*	*
*	A_2	*
*	*	*

$\longrightarrow \boxed{P}$ (1)[100%]

*	*	*
*	A_2	*
*	*	*

$\longrightarrow \boxed{I}$ (1)[100%]

Cette fois on impose une probabilité d'application des trois règles de 100% car on veut que A_2 change à coup sûr d'un jour à son lendemain. Et de même pour A_3 ... à vous de compléter sur les mêmes idées.

Remarque : *espace « infini »*
Il n'est pas rare que l'on veuille modéliser de grands espaces, contrairement à une petite classe comme dans l'exemple précédent (par exemple pour modéliser des mouvements de populations, de troupeaux, *etc*). Prendre une très grande surface finie (e.g. un très grand rectangle) semble une idée naturelle, mais elle est très mauvaise pour deux raisons :

— D'une part, il faudrait utiliser énormément de mémoire pour mémoriser l'état de l'automate.
— D'autre part, aussi grand que soit l'espace modélisé, il aurait des bords et les évolutions modélisées au voisinage de ces bords induirait des artefacts. Par exemple sur l'étude de migrations, il induirait des accumulations d'individus ou des changements de direction non crédibles.
On pourrait penser qu'il suffit de ne regarder que ce qui se passe « au centre » pour observer un comportement sans artefacts. Il n'en est rien ! En fait, les comportements aberrants des bord influent très rapidement le comportement de toutes les zones de l'espace.

Pour contourner ces deux écueils, on replie un espace carré, de taille raisonnable, en un tore et on le « peuple » pour respecter les densités spatiales. De manière empirique, on remarque que l'absence de bords dans le tore simule remarquablement bien un espace infini.

3 Automates synchrone, non probabiliste : TD/interactions moléculaires

Pour bien comprendre les limitations des diverses versions des automates cellulaires, on va modéliser au moyen d'un *automate cellulaire 2D synchrone de taille 5×5* une portion d'ADN dans un noyau, en supposant qu'elle est soumise à un flux de facteurs de transcription allant de la gauche vers la droite. L'état de départ que nous ferons évoluer sera le suivant :

État E_0 =

t	-	-	g_1	-
t	-	g_2	-	-
t	-	g_1	-	-
t	g_2	-	-	-
-	-	g_1	-	-

Par convention, un tiret « - » dans une case signifie que la case est « vide », la lettre « t » représente un facteur de transcription, enfin la portion d'ADN est symbolisée par la ligne alternée des deux gènes « g_1 » et « g_2 », qui sont respectivement en 3 et 2 exemplaires dans cette portion d'ADN. On suppose que g_1 et g_2 utilisent le même facteur de transcription t.

Le flux des facteurs de transcription est modélisé par deux règles : si la case à gauche d'une case vide (-) est un facteur de transcription (t), alors la case vide se remplit du facteur de transcription ; si la case à droite du facteur de transcription est vide, alors la case du facteur de transcription devient vide.

$flux_1$:

*	*	*
t	-	*
*	*	*

→ | t |

$flux_2$:

*	*	*
*	t	-
*	*	*

→ | - |

Rappelons que dans la partie gauche d'une règle, une case qui contient une étoile signifie que la règle peut s'appliquer quel que soit le contenu de cette case. Par ailleurs, par convention, on traitera les bords comme s'il étaient entourés d'une membrane formée de molécules inertes « m » dans des cases tout autour de nos 5×5 cases ; ainsi nos deux règles s'appliquent aussi sur les bords haut et bas, la première s'applique sur le bord droit mais pas la seconde, et inversement avec le bord gauche.

Lorsqu'un facteur de transcription rencontre un gène à sa droite, il se fixe dessus, fournissant un gène en cours de transcription noté « gt » :

$fixation_1$:

*	*	*
t	g_1	*
*	*	*

→ | gt_1 |

$fixation_2$:

*	*	*
*	t	g_1
*	*	*

→ | - |

$fixation_3$:

*	*	*
t	g_2	*
*	*	*

→ | gt_2 |

$fixation_4$:

*	*	*
*	t	g_2
*	*	*

→ | - |

Un gène en cours de transcription produit de l'ARN, toujours selon un flux de gauche à droite. On note « a_1 » et « a_2 » les ARN des gènes g_1 et g_2 respectivement :

$transcrit_1$:

*	*	*
gt_1	-	*
*	*	*

→ | a_1 |

$transcrit_2$:

*	*	*
gt_2	-	*
*	*	*

→ | a_2 |

Un gène en cours de transcription peut également perdre son facteur de transcription. On supposera que t s'échappe

en dessous du gène, sous réserve qu'il en ait la place, c'est-à-dire que la case d'en dessous soit vide et ne puisse pas être remplie par de l'ARN ou un autre facteur de transcription :

$dissoc_1$:

*	*	*
*	gt_1	*
$g_1 \vee g_2 \vee$-	-	*

$\rightarrow$ g_1

$dissoc_2$:

*	*	*
*	gt_2	*
$g_1 \vee g_2 \vee$-	-	*

$\rightarrow$ g_2

$dissoc_3$:

*	$gt_1 \vee gt_2$	*
$g_1 \vee g_2 \vee$-	-	*
*	*	*

$\rightarrow$ t

Par convention, « $g_1 \vee g_2 \vee$- » signifie que la case peut être remplie par g_1 **ou** par g_2 **ou** être une case vide. La même convention s'applique pour « $gt_1 \vee gt_2$ ». Ainsi les règles $dissoc_1$ et $dissoc_2$ représentent en réalité 3 règles chacune, et la règle $dissoc_3$ en représente 6.

Exercice 6 : Écrivez selon le principe ci-dessus trois règles $migre_1$, $migre_2$ et $migre_3$ qui modélisent comme pour le facteur de transcription un flux de gauche à droite de l'ARN. On pourra utiliser une notation condensée comme pour les 3 règles *dissoc*.

Exercice 7 : Écrivez les états successifs de l'automate cellulaire synchrone, en partant de l'état E_0 et en appliquant toutes les règles ($flux$, $fixation$, $transcrit$, $dissoc$ et $migre$) jusqu'à ce que plus aucune règle ne s'applique.

Exercice 8 : Supposons que l'on veuille ajouter la possibilité d'une migration vers le bas de l'ARN « a_2 ». Expliquez pourquoi les règles « naïves » ci-dessous sont incohérentes avec les autres règles :

$migre_4$:

*	a_2	*
*	-	*
*	*	*

$\rightarrow$ a_2

$migre_5$:

*	*	*
*	a_2	*
*	-	*

$\rightarrow$ -

INDICATION : tester par exemple ces règles avec les états construits à l'exercice précédent.

Exercice 9 : Donnez la liste des règles avec lesquelles $migre_4$ ou $migre_5$ est en désaccord. Proposez ensuite des règles qui effectuent la migration vers le bas de a_2 uniquement quand elle est nécessairement cohérente.

Exercice 10 : Supposons maintenant que l'on veuille que les ARN puissent migrer dans toutes les directions. Proposez des tentatives de règles pour modéliser cette migration tout en préservant les quantités de matières. Pour cela, on s'autorisera des voisinages de rayon supérieur à 1 et l'on pourra modifier le cas échéant les règles déjà écrites de migration de l'ARN.

Conclusion : les règles des automates cellulaires rendent extrêmement difficile, et même impossible sans créer d'artefacts, de modéliser à la fois des phénomènes *isotropes* et une *conservation de la matière*. Et cela quelle que soit la variante choisie !

4 Automates asynchrones stochastiques à base de règles

La conclusion précédente conduit à considérer une extension des automates cellulaires stochastiques dans laquelle une *règle* aura pour objet de mettre à jour plusieurs cases en même temps afin d'assurer une conservation globale de la matière.

On considère alors des règles « sans les directions des cases », de la forme $a_1 + \cdots + a_n \rightarrow b_1 + \cdots + b_n$ [p] où :

- n est inférieur ou égal au nombre maximum de cases qui peuvent être voisines 2 à 2 (max=4 avec le voisinage à 9 cases dans le plan, max=2 pour le 1D et pour le voisinage en croix en 2D et 3D, max=8 en 3D avec toutes les diagonales possibles pour le voisinage).
- La dynamique est asynchrone en ce sens qu'on ne choisit (aléatoirement) qu'un seul ensemble de n cases voisines à la fois mais elle est d'une certaine façon bloc-synchrone en ce sens que les n cases choisies sont mises à jour en même temps.
- Chacun des a_i est remplacé par le b_i correspondant.
- Un poids (P) n'aurait pas beaucoup d'intérêt en pratique car on choisit d'abord aléatoirement une case, puis les directions utiles pour la partie gauche des règles : dès lors, une approximation fréquente est d'intégrer le poids P de choix de règle dans répartition des probabilités d'application p. Cela induit la contrainte que, si plusieurs règles peuvent s'appliquer pour un état donné, il faut s'assurer que la somme des probabilités ne dépasse jamais 1.

Cette nouvelle façon de gérer les mises à jour résout clairement les problèmes de conservation de la matière : il suffit de n'écrire que des règles qui respectent cette conservation de la matière et, contrairement aux automates cellulaires, cette propriété « locale » est préservée globalement.

Reprenons l'exemple des 2 gènes avec un facteur de transcription commun. La version à base de règles permet d'avoir une diffusion non directionnelle, de conserver la matière, de favoriser l'un des gènes si l'on veut, *etc.*
— $t + \text{-} \rightarrow \text{-} + t$ et $a_1 + \text{-} \rightarrow \text{-} + a_1$ et $a_2 + \text{-} \rightarrow \text{-} + a_2$ (p proportionnels aux vitesses de diffusion respectives)
— $t + g_1 \rightarrow \text{-} + gt_1$ et $t + g_2 \rightarrow \text{-} + gt_2$ (p proportionnels aux affinité)
— $gt_1 + \text{-} \rightarrow g_1 + t$ et $gt_2 + \text{-} \rightarrow g_2 + t$ (p proportionnels aux fréquences de détachement))
— $gt_1 + \text{-} \rightarrow gt_1 + a_1$ et $gt_2 + \text{-} \rightarrow gt_2 + a_2$ (p proportionnels aux vitesses de transcription)
Cependant les automates asynchrones stochastiques à base de règles ne constituent pas pour autant une solution de modélisation universelle...

5 Tentative de version à base de règles du feu de forêt...

On veut cette fois utiliser des automates asynchrones stochastiques à base de règles pour modéliser la propagation du feu en forêt, avec les mêmes notations que dans le cours précédent.

Exercice 11 : Écrivez des règles qui traduisent les deux idées suivantes :
- Un arbre sain, lorsqu'il est atteint par le feu, brûle *en moyenne* pendant 3 de ses mises à jour avant de devenir calciné.
- Un arbre sain ayant un arbre en feu dans son voisinage a une chance sur deux de prendre feu lors d'une mise à jour. (Nota : ne pas oublier que « le temps passe » aussi pendant cette mise à jour).

Une solution possible : on remplace les *f1* à *f3* par *f* simplement. De nombreuses interprétations sont possibles, c'est une question de choix de conception du modélisateur. Par exemple on peut proposer :
— $\text{f} \rightarrow \text{c}$ $[\frac{1}{3}]$ [en fait $\frac{1}{3}$ n'est qu'une approximation, les modèles de durée de vie sont complexes]
— $\text{s} + \text{f} \rightarrow \text{f} + \text{f}$ $(\frac{1}{3})$
— $\text{s} + \text{f} \rightarrow \text{f} + \text{c}$ $(\frac{1}{6})$
— $\text{s} + \text{f} \rightarrow \text{s} + \text{f}$ $(\frac{1}{3})$
— $\text{s} + \text{f} \rightarrow \text{s} + \text{c}$ $(\frac{1}{6})$
Entre autres remarques, les deux dernières règles ne sont pas absolument nécessaires car on peut considérer qu'une règle non appliquée est un coup dans l'eau qui ne fait pas passer le temps. La troisième est prototypique et mérite commentaires : toutes ces règles sont en fait des choix de modélisation à peser lorsqu'on construit une simulation.

On peut critiquer cette « solution » sous de nombreux angles. Par exemple la probabilité de prendre feu devrait augmenter avec le nombre de voisins en feu. Si l'on tente d'écrire des règles qui traduisent cette idée, on échoue pour au moins trois raisons :
— ces règles ne sont pas centrées sur le devenir d'une case en particulier, il faut donc considérer le devenir de toutes les entités du membre gauche de chaque règle,
— les voisinages sont partiels (limités à 4 cases dans le plan au lieu de 9 parce que toutes les cases doivent être voisines les unes des autres),
— à supposer qu'on ait assez de cases dans le voisinage, les probabilités/poids sont très difficiles à calculer...

6 Au-delà des automates cellulaires

Si l'on adopte un formalisme à base de règles, finalement plus rien n'impose de découper l'espace en cases comme dans les automates cellulaires. En effet, il suffit que l'on définisse dans quelles conditions on considère que les a_i des règles $a_1 + \cdots + a_n \rightarrow b_1 + \cdots + b_n$ [p] sont voisins les uns des autres et une règle sera applicable uniquement si tous les a_i sont voisins deux à deux.

Cette approche est beaucoup plus souple que les automates cellulaires. Par exemple elle n'impose pas de nombre maximal de réactants, elle n'impose pas que les a_i soient à des emplacement de forme et de taille fixées à l'avance, *etc.* On peut pas exemple donner un rayon à chaque molécule qui dépend de la nature de chacun des a_i, donner une position quelconque dans l'espace de toutes les molécules, et considérer que deux molécules sont voisines si leur distance est inférieure à la somme de leurs rayons.

Une telle approche conduit rapidement à considérer que chaque molécule est un *objet* qui possède en propre sa position dans l'espace, qui partage son rayon avec les molécules de même nature,... et on en arrive vite à penser en terme de programmation dite « Orientée Objets », et encore au-delà à un style de modélisation dit « multi-agent ».

6.1 Survol des principes de la programmation objet

Lorsqu'on écrit de gros logiciels, le nombre de variables globales peut devenir très important. Lorsque plusieurs programmeurs participent au développement (ou même lorsque la taille du logiciel ne permet pas à un seul programmeur d'avoir toutes ces variables en tête) le risque qu'un même nom de variable soit utilisé pour deux choses très différentes devient très grand. On a alors besoin de langages qui permettent d'*encapsuler* les noms de variables de telle sorte qu'une variable n'est visible et utilisable que dans un certain contexte. Dans ce cas, un même nom de variable dans deux contextes différents ne dénote en réalité pas la même variable, ce qui évite les interactions non souhaitées.

La programmation orientée objet (POO) et les langages orientés objet (LOO) comme C++, java, python, *etc.* utilisent la notion d'*objet* pour mettre en œuvre cette encapsulation. Un objet « possède » des variables qu'il est le seul à voir et à pouvoir en changer la valeur. Ces variables « privées » sont appelées les *attributs* de l'objet. On peut alors modifier l'état d'un objet uniquement au travers d'une interface qui est constituée d'un ensemble de fonctions que l'objet peut faire tourner. Ces fonctions sont appelées des *méthodes* et elles sont exécutées en indiquant non seulement le nom de la méthode mais également le nom de l'objet qui doit l'exécuter. Une interface, c'est-à-dire l'ensemble des méthodes qu'un objet peut exécuter, caractérise donc un type d'objets et plusieurs objets de même type peuvent être présents dans un logiciel.

Vu d'un programmeur qui *utilise* un objet, il n'a aucune connaissance ni des attributs qui composent l'objet, ni du code qui réalise chaque méthode. Il dispose simplement d'une spécification de ce que fait chaque méthode de l'interface. Les attributs et la manière dont chaque type d'objet réalise ses méthodes constitue ce qu'on appelle une *classe* d'objets. Tous les objets d'une même classe sont donc des « clones » (instances de leur classe) pour ce qui concerne ces attributs et méthodes communs.

Souvent, des classes d'objets peuvent être obtenues en enrichissant une classe d'objets déjà existante (plus d'attributs ou plus de méthodes, tout en restant cohérent avec ceux de la classe d'origine, ou au contraire une restriction de l'ensemble des valeurs que peuvent prendre certains attributs). Pour éviter des redites, les LOO offrent des mécanismes d'*héritage* dont nous ne parlerons pas ici...

6.2 Survol des principes des systèmes multi-agents

Pour des simulations plus élaborées que des interactions moléculaires, les automates, qu'ils soient cellulaires ou à base de règles, peuvent rapidement devenir très compliqués à mettre en œuvre, avec de nombreux états (le nombre de journées de feu d'un arbre par exemple montre comment la multiplication des noms d'états peut conduire à un nombre croissant d'identifiants). On a rapidement envie de mettre un objet avec des attributs dans chaque case de l'automate... De plus, puisque les objets disposent de méthodes, on pourrait déléguer aux objets eux-mêmes, par des méthodes d'exploration de leur voisinage, la tâche de s'appliquer à eux-mêmes les règles de transformation locales. Pour des simulations très élaborées, on pourrait même écrire des méthodes qui mettent à jour un ensemble d'attributs très riche et du coup peuvent « apprendre » au cours du temps par accumulation/modification de « connaissances » et modifient leur comportement en fonction de « l'expérience passée » de l'objet tout au long de la simulation. C'est par exemple le cas pour les simulations de réseaux de neurones du cerveau.

On en arrive vite à considérer que ces « objets » ont une autonomie propre lors des simulations et on parle alors de simulations par *Système multi-agents* (SMA). Cela conduit à un type de conception des simulations où plusieurs questions sont systématiquement abordées :

- Il y a un espace dans lequel évoluent les agents et par conséquent il faut aussi modéliser un *environnement* dont les caractéristiques sont accessibles à tous les agents, du moins celles qui concernent le voisinage dans lequel ils se trouvent. Les agents peuvent modifier le comportement de leurs actions (≈méthodes) en fonction de cet environnement. Il faut donc définir leur capacité de perception de cet environnement en accord avec le système que l'on simule. Ils peuvent aussi modifier cet environnement (e.g. des fourmis qui déposent des phéromones).
- Il faut organiser l'état interne d'un agent (≈attributs) pour y programmer des règles de comportement et même des capacités d'apprentissage de nouvelles règles de comportement ou de modifications de ces règles et de leurs paramètres.
- Chaque agent doit pouvoir communiquer avec les autres agents (≈déclencher certaines de leurs méthodes) selon des critères et des conditions en accord avec le système que l'on simule. On peut même en arriver à concevoir des mini-langages de communication entre agents.

L'avantage des SMA est que, comme on peut sophistiquer à volonté chaque agent, il est possible de se rapprocher des particularités des éléments biologiques considérés avec autant de détail que l'on veut. Ceci offre une meilleure crédibilité biologique des simulations qui sont effectuées que dans le cas des automates « simples ». En revanche, le défaut majeur est que cette complexité du modèle informatique interdit tout raisonnement sur le système lui même : on observe des phénomènes émergeants lors des simulations mais, même si ceux-ci se répètent à toutes les simulations,

on n'a aucune certitude que ce soient des propriétés générales et la simulation $n + 1$ peut toujours contredire toutes les précédentes. Pire, il est parfois aussi difficile de comprendre ce qui se passe dans un SMA que dans la réalité du modèle biologique.

Pour débuter dans la programmation multi-agents, le langage et l'environnement logiciel NetLogo est un bon début, lorsque l'espace à modéliser est planaire. C'est un logiciel open-source et libre :

`https://ccl.northwestern.edu/netlogo/`