Stratégie d'application stochastique de règles de réécritures dans le langage MGS

Antoine Spicher & Olivier Michel

Laboratoire de Méthodes Informatiques,
523 place des terrasses de l'agora
91000 ÉVRY CEDEX, France
{aspicher, michel}@lami.univ-evry.fr

L'objectif du projet MGS est l'étude et le développement d'un langage de programmation dédié à la modélisation et à la simulation de phénomènes dont la structure est dynamique. Les langages dédiés, souvent déclaratifs et organisés autour d'un petit noyau, proposent de nouvelles abstractions et notations orientées vers un domaine d'application particulier, rendant la prise en compte des phénomènes modélisés plus concise et la réutilisation des codes plus aisée.

Les processus auxquels nous nous intéressons sont des systèmes hautement structurés et hiérarchisés. La spécification de tels systèmes passe souvent par la caractérisation d'un état décrit par l'association d'attributs (volume, masse, température, charge, etc.) à une structure spatiale.

L'idée fondamentale de MGS est de permettre la spécification de ces états au moyen d'une nouvelle structure de données [GM02a] fondée sur l'organisation topologique des systèmes modélisés. Ainsi, la fonction d'évolution de ces systèmes correspond à une fonction qui transforme un état du système en un autre. La spécification de ces fonctions d'évolution est simplifiée par l'utilisation d'une définition par cas reposant sur un langage de filtre puissant. MGS peut être considéré comme un langage fonctionnel standard étendu d'une nouvelle structure de données, les *collections topologiques*, et d'un nouveau mécanisme de définition de fonction, les *transformations*. Un point de vue alternatif correspond à interpréter les transformations comme l'application de règles spécifiant l'évolution locale d'un état vu comme une structure de données complexe. Il apparaît que ces notions développées pour la simulation de systèmes dynamiques à structure dynamique (également appelés $(\text{SD})^2$) nous conduisent vers un nouveau style de programmation permettant une description concise de plusieurs algorithmes fondamentaux sur les ensembles, les séquences, les graphes, etc.

L'introduction présente le cadre dans lequel se situe nos travaux, celui de la réécriture de structure de données plus riches que les types de données algébriques. Puis, après une brève présentation du langage MGS, nous nous intéresserons aux problèmes liés aux différentes stratégies d'application des règles au sein d'une transformation MGS. En effet, la stratégie *parallèle maximale* par défaut de MGS est parfaitement adaptée aux systèmes dynamiques décrits par un modèle *synchrone*. Nous présentons une nouvelle classe de stratégies *asynchrones* qui permettent la prise en compte de processus *stochastiques*. Nous illustrerons par plusieurs exemples, dans le domaine général de l'algorithmique probabiliste (*random algorithms*) et dans le domaine plus particulier de la biologie cellulaire avec la modélisation du processus d'infection d'une bactérie par le bactériophage lambda.

1. Introduction

MGS, en tant que langage dédié, se concentre sur la modélisation et la simulation de systèmes biologiques. Les systèmes biologiques font partie des systèmes difficiles à modéliser. Cette difficulté provient de la grande complexité des mécanismes qui entrent en jeu et de l'aspect fortement dynamique et concurrent de ces systèmes.

L'expression locale des règles d'évolutions des systèmes biologiques couplée aux modifications de la structure même des systèmes décrits (on parle alors de *systèmes dynamiques à structure dynamique* ou $(SD)^2$ [GGMP02], c'est-à-dire dont la structure évolue au cours du temps) rend la description et la manipulation de systèmes biologiques difficiles. Nous proposons, pour cela, d'utiliser des techniques de réécriture sur des structures de données complexes et dynamiques.

En effet, les systèmes de réécriture (SR), initialement développés pour formaliser le raisonnement équationnel, sont aussi utilisés pour exprimer les changements d'état d'un processus. La capacité des système de réécriture à spécifier le remplacement d'une sous-partie d'un objet par un autre semble désigner les SR comme un formalisme adapté à rendre compte de lois d'évolution *locales*. La simulation d'un système biologique se rapproche alors de l'application itérée (selon des stratégies diverses) des règles d'un système de réécriture sur une structure de données représentant l'état courant du système.

Pour augmenter le pouvoir d'expressivité des systèmes de réécriture (c'est-à-dire aller au delà des termes qui ne permettent que la représentation d'organisations arborescentes), nous avons développé de nouvelles techniques de réécriture appliquées sur des structures de données plus riches, *les collections topologiques*.

Le point de vue topologique adopté considère une structure de données comme un ensemble d'éléments organisés suivant une relation de voisinage qui définit quels éléments de la structure sont accessibles à partir d'un élément donné. Les *transformations*, suivant un point de vue local, sont des fonctions permettant la manipulation de collections topologiques spécifiant des règles de réécriture reposant sur cette relation de voisinage topologique.

L'objectif du projet MGS est d'intégrer le formalisme des systèmes de réécritures dans un langage de programmation dédié à la modélisation et la simulation de $(SD)^2$. Nous présentons le langage dans cette section. MGS consiste en un langage fonctionnel, complet, impur, typé dynamiquement et strict. Nous détaillons dans la section suivante les collections topologiques, principale structure de données du langage MGS.

2. Les collections topologiques

Une des particularités du langage MGS est sa capacité à manipuler des données structurées par une *topologie abstraite* à travers des *transformations* [GM02c]. La notion de structure de données est unifiée dans la notion de *collection topologique*, un ensemble d'entités organisées par une topologie abstraite.

2.1. Topologie abstraite combinatoire

Les collections topologiques fournissent un point de vue des structures de données où l'agencement des éléments les uns avec les autres est mis en avant. La définition d'une collection topologique requiert une structure (c'est-à-dire un support décrivant l'organisation des entités) qui sera ensuite décorée par des données.

Pour décrire une organisation plus complexe qu'un tableau (nous avons déjà montré dans [GM02b, GMC02, GMS95] une généralisation de la notion de tableaux) ou qu'une structure arborescente, nous avons choisi d'utiliser une notion fondamentale en *topologie algébrique combinatoire*. La structure d'une collection topologique passe par la spécification d'un espace topologique nommé *complexe cellulaire abstrait* [Ale82, Hen94]. Un complexe est un ensemble de *cellules topologiques* de différentes dimensions, encore appelée n-cellules (où n est la dimension de la cellule). Chaque cellule est en fait l'abstraction d'un espace de même dimension : les 0-cellules sont des points, les 1-cellules des arcs, les 2-cellules

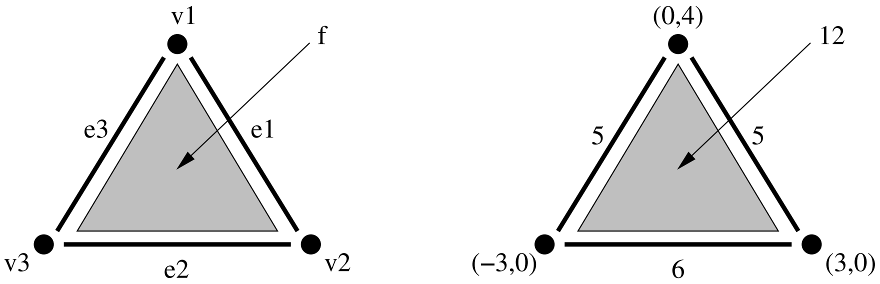


FIG. 1 – La figure de gauche représente un complexe cellulaire composé de trois 0-cellules (v_1, v_2, v_3), de trois 1-cellules (e_1, e_2, e_3), et d'une 2-cellule f. Le bord de f est constitué de ses cellules incidentes v_1, v_2, v_3, e_1, e_2 et e_3. Plus particulièrement, les trois arcs sont les faces de f, et par conséquent, f est une coface de e_1, e_2 et e_3. Sur la figure de droite des données sont associées aux cellules topologiques : des positions pour les sommets, des longueurs pour les arcs et une aire pour f.

des surfaces, les 3-cellules des volumes, etc. Ces briques sont ensuite agencées les unes par rapport aux autres pour former un complexe suivant une *relation d'incidence* reposant sur la notion de bord : soient c_1 et c_2, respectivement, une n_1-cellule et une n_2-cellule, telles que $n_1 < n_2$. On dira que c_1 est incidente à c_2 si c_1 appartient au bord de c_2. Si $n_1 = n_2 - 1$, c_1 est dite *face* de c_2 et c_2 est une *coface* de c_1. De plus, pour chaque cellule c d'un complexe C, si c' est une face de c, alors c' appartient également à C. Cette structure généralise l'idée de graphe, qui est en fait un complexe composé uniquement de 0-cellules et de 1-cellules.

On décore de différentes valeurs chaque cellule topologique. Ceci correspond au concept de *chaîne topologique* en topologie algébrique, et nous apporte une définition formelle des collections topologiques. Nous ne détaillerons pas plus ces notions dans cet article, mais le lecteur intéressé peut approfondir ce sujet avec [Mun84, GM02c]. La figure 1 donne un exemple de collection topologique.

2.2. Types de collection

La relation d'incidence peut satisfaire certaines propriétés qui permettent de redéfinir les structures de données standards comme des collections topologiques. La plupart d'entre elles, correspondent à un complexe cellulaire de dimension 1 (c'est-à-dire dont les cellules sont de dimension inférieure ou égale à 1, autrement dit des graphes), où seuls les sommets sont décorés. On peut ainsi redéfinir entre autres la séquence en tant que graphe linéaire ; l'ensemble (et le multi-ensemble) comme un complexe où chaque sommet est connecté à tous les autres par un arc ; un tableau qui correspond à un maillage régulier ; ...

Ainsi, chaque type de collection définit une relation de voisinage. Celle-ci est utilisée pour induire la notion de *sous-collection*. Une sous-collection S' d'une collection S est un sous-complexe de S dont la relation d'incidence vérifie les mêmes propriétés que celle de S. Autrement dit, S' hérite de l'organisation topologique de S.

2.3. Collection topologique et représentation de l'état d'un système dynamique

Les collections topologiques permettent de représenter facilement les états complexes d'un système dynamique à un instant donné. Les éléments de la collection topologique sont les éléments atomiques du système dynamique, chaque élément étant décoré par une valeur. On associe également à chaque élément sa *position* dans la collection, qui n'est autre que la cellule topologique qu'elle décore, afin de savoir où il se situe exactement dans la collection. Dans la majeure partie des cas, il n'est pas nécessaire de distinguer la position de sa valeur associée. Dans ce cas, on utilise l'expression « élément de la collection ». Dans ce contexte, la relation de voisinage rend compte des interactions possibles entre les éléments de la collection [SMG04].

3. Transformations

Les collections topologiques représentent un cadre possible pour l'extension des SR. En effet, la relation de voisinage fournit une vue locale de l'organisation structurelle des éléments. Les *transformations* étendent la notion de SR vers la prise en compte de structures différentes (et plus riches) que les arbres et permettent de spécifier la fonction d'évolution du système dynamique modélisé. La *transformation* d'une collection topologique S consiste en l'*application parallèle* d'un ensemble de *règles de réécritures locales*. Une règle de réécriture locale r spécifie le remplacement d'une sous-collection par une autre sous-collection. L'application d'une règle de réécriture $\sigma \Rightarrow f(\sigma, ...)$ à une collection S :

1. sélectionne une sous-collection S_i de S dont les éléments filtrent le *motif* σ,
2. calcule une nouvelle collection S'_i comme le résultat de l'application de la fonction f de S_i et de ses voisins,
3. spécifie l'insertion de S'_i en lieu et place de S_i dans cs.

Un motif σ dans la partie gauche d'une règle spécifie une sous-collection où une interaction à lieu. Cette sous-collection peut avoir une forme arbitraire, rendant sa spécification complexe. Il existe deux façons de spécifier une sous-collection en MGS. La première, appelée *patch*, dont nous ne parlerons pas ici est générale et explicite.

Comme nous travaillons en général sur des structures de données réductibles à un graphe où seul les sommets sont valués, un autre langage de filtres a été développé et est dédié à cette classe de collections topologiques. Il est fondé sur l'énumération séquentielle d'éléments ; on entend par « séquentielle » le fait que deux éléments contigus dans cette énumération sont *voisins*, c'est-à-dire qu'ils sont reliés par un arc. Une telle énumération est appelée un *chemin*. La figure 2 décrit un exemple de sous-collection définie par un chemin. Dans les paragraphes suivants, nous ne décrivons que le langage de filtre de chemin (les patchs n'étant pas utilisés dans la suite de l'article).

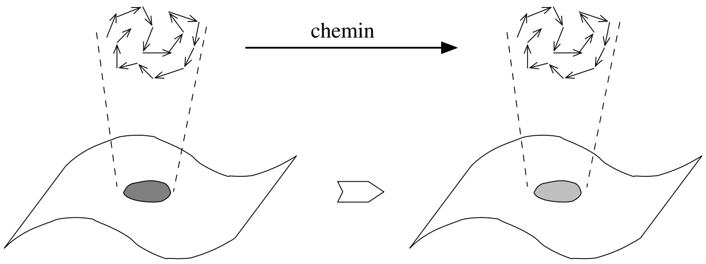


Fig. 2 – La spécification de sous-collection dont la forme est arbitraire requiert la spécification d'un *chemin* qui filtre des séquences d'éléments voisins deux à deux ; bien qu'ils ne permettent pas de représenter n'importe quelle forme de sous-collection, leur spécification est concise et particulièrement expressive.

3.1. Motif de chemin

Un motif de chemin `Pat` est une séquence ou la répétition `Rep` d'un motif de filtrage de base. Un motif de filtrage `BF` filtre un élément. Le fragment de grammaire des motifs de chemins reflète cette décomposition :

$$\mathtt{Pat} ::= \mathtt{Rep} \mid \mathtt{Rep}\ \mathtt{,}\ \mathtt{Pat} \qquad \mathtt{Rep} ::= \mathtt{BF} \mid \mathtt{BF}/exp \qquad \mathtt{BF} ::= \mathrm{id}$$

où `cte` est une valeur littérale, id est une variable de motif et *exp* une expression booléenne. L'explication suivante donne une interprétation systématique pour ces motifs :

variable : une variable de motif a filtre exactement un élément avec une valeur déterminée. La variable a peut apparaître n'importe où dans les gardes du reste de la règle et dénote la valeur de l'élément filtré.

voisinage : b`,` p est un motif qui filtre un chemin débutant par un élément filtré par b et qui se poursuit par un chemin filtré par p, le premier élément de p étant voisin de b (ils sont donc liés par un arc incident commun).

garde : p/exp filtre un chemin filtré par p si le prédicat exp est vrai.

Voici un exemple de motif : x`,` y `/` y `>` x. Il permet de filtrer deux éléments voisins (le voisinage est spécifié par la virgule), nommés `x` et `y`, tels que la valeur associée à y est plus grand que la valeur associée à x. Le langage de filtres de chemins est inspiré par la grammaire des expressions régulière et d'autres constructions comme l'itération sont également disponibles [GMC02].

3.2. Remplacement

La partie droite d'une règle spécifie une collection qui remplace la sous-collection filtrée par le motif donné en partie gauche de la règle. Là encore, la spécification d'une telle collection, n'est pas évidente.

Cependant, il existe un point de vue alternatif dans le cas du filtrage de chemin : la succession des motifs décrivant une séquence d'éléments, la partie droite d'une règle peut être une expression qui s'évalue également en une séquence d'éléments. La substitution se fait alors élément par élément : l'élément i dans le motif filtré est remplacé par le i^{e} élément de la partie droite de la règle. Ce point de vue permet une écriture extrêmement concise des règles.

Cette spécification du remplacement ne semble pas autoriser une modification de la topologie, la séquence réécrite devant être de même longueur que le chemin filtré. Or pour certains types de collection, il est possible de remplacer un chemin filtré par une séquence dont la longueur est différente. On les appelle des collections *leibniziennes* et sont opposées aux collections *newtoniennes*. Leur capacité à accepter un changement de topologie, vient du fait que le voisinage est calculé en fonction des éléments qu'elles contiennent. On peut citer les ensembles, les multi-ensembles (ensembles autorisant plusieurs occurrences d'un même élément), les séquences, les graphes arbitraires, ... En revanche, une grille bidimensionnelle est un exemple de collection newtonienne : on ne peut pas remplacer un sous-ensemble quelconque d'une grille par une autre forme sans détruire le voisinage 2D de la grille.

3.3. Stratégie d'application des règles

MGS dispose d'une stratégie d'application des règles par défaut : la stratégie synchrone. Il en existe effectivement trois différentes selon l'ordre dans lequel les règles sont appliquées. Toutes les trois partagent la même propriété; elles sont *maximales parallèles* :

maximale Les sous-collections qui sont filtrées lors de l'application d'une transformation sont telles qu'il n'existe plus de sous-collection pouvant être filtrée par aucune des règles dans les parties non filtrées de la collection.

parallèle Toutes les sous-collections filtrées sont réécrites en parallèle. Bien entendu, cela est possible car toutes les sous-collections filtrées sont indépendantes les unes des autres et que leur intersection deux à deux est vide.

Pour l'ordre d'application des règles, on peut choisir de filtrer (1) en donnant une priorité plus forte aux premières par rapport aux suivantes, ou (2) en laissant l'interprète fixer cette priorité de façon aléatoire c'est-à-dire en ne tenant pas compte de l'ordre de définition des règles, ou enfin (3) ne demandant aucune priorité entre les règles (elles sont donc appliquées de façon totalement aléatoire).

Par défaut, le prototype du langage MGS effectue un filtrage maximal parallèle avec priorité sur les règles, en accord avec l'approche adoptée dans les systèmes de Lindenmayer [LJ92] qui permettent de faire de la réécriture maximale parallèle de chaînes (*string rewriting*).

Nous avons développé une nouvelle stratégie pour l'application des règles au sein d'une transformation. En effet, la stratégie synchrone décrite ci-dessus ne permet pas de rendre compte de systèmes dynamiques décrits par des modèles stochastiques. Nous présentons dans la section suivante cette nouvelle stratégie.

4. Stratégie d'application stochastique des règles

La stratégie par défaut d'application des règles de MGS est parfaitement adaptée à la description de processus synchrones. On souhaite étendre la classe de processus descriptible par MGS en intégrant une nouvelle stratégie de réécriture, asynchrone et probabiliste. En effet, un système de réécriture avec une stratégie *stochastique* d'application des règles permet de rendre compte d'un grand nombre d'applications qui ne sont actuellement pas modélisables en MGS (ou alors au prix de constructions complexes). Citons par exemple les algorithmes aléatoires [MR99], la formalisation et l'analyse de protocoles [AGG$^+$05], les systèmes objets distribués [KSMA03], les algorithmes génétiques [Gol89], les bases de données probabilistes [ELLS01], ...

Parallèlement à ce souhait, on peut constater que de nombreux travaux plus théoriques sur les systèmes de réécriture se proposent d'intégrer, à divers niveau, des mécanismes permettant l'application probabiliste des règles. Citons par exemple les travaux autour de Maude [KKMA03], où les auteurs définissent la notion de théories de réécritures *probabilistes* qui donnera lieu à la définition d'un prototype probabiliste, PMaude ; de même, le système Elan se voit doter dans [BK02] d'une stratégie de réécriture probabiliste.

Mais notre motivation initiale est d'intégrer simplement dans un système de réécriture l'algorithme *Stochastic Simulation Algorithm* (SSA) défini par Gillespie [Gil77] dans les années 70 qui permet de modéliser et d'effectuer des simulations *exactes* (au sens donné par Gillespie) de processus biochimiques. Nous verrons dans la section suivante que l'intégration en MGS d'une stratégie stochastique ne nécessite que peu de changements pour prendre en compte cette méthode.

4.1. Rappel sur la stratégie maximale parallèle

Une transformation MGS T est la donnée d'un ensemble de règles R_i ainsi que d'arguments optionnels permettant de contrôler l'application de *toutes* les règles. Chaque règle $n_i : m_{i1}, ..., m_{ij} = \{o_{i1}, ..., o_{ik}\} => d_{i1}, ..., d_{il}$ est composée d'un nom n_i (optionnel), d'un motif de chemin m_{ij} (décrit à la section 3.1), d'options o_{ik} permettant de contrôler l'application de la règle R_i considérée et d'une partie droite d_{il} spécifiant les valeurs à substituer aux valeurs filtrées selon m_i : soit f une fonction d'arité j, on a $d_{i1..l} = f(m_i1, ..., m_ij)$ (où la relation qui lie l et j dépend de la nature de la collection sur laquelle est appliquée T, voir la section 3.2).

Par exemple, une transformation standard T est spécifiée de la façon suivante (`selection` est le nom donné à la seule règle définie) :

```
T =
{
  selection : x, y / (x == (2 * y)) => f(x, y)
}
```

Une trace possible d'une application de T sur le multi-ensemble (un multi-ensemble est une collection topologique qui correspond à un graphe complet où chaque sommet est voisin de tous les autres) $(2, 4, 3, 7, 2, 6)$ avec $f(x, y) = x + y$ est :

1. instanciation des motifs $\underbrace{2,\quad 4}_{x\equiv 4 \quad y\equiv 2\ \Rightarrow 6},\quad \underbrace{3,\quad 6}_{x\equiv 6 \quad y\equiv 3\ \Rightarrow 9},\quad 7,\quad 2$
2. calcul de la collection résultat (6, 9, 7, 2)

La stratégie *maximale parallèle* par défaut (Cf. la section 3.3) sélectionne pour chaque R_i *toutes* les instances *indépendantes* du motif m_i dans la collection topologique et remplace ces instances par le résultat de la partie droite de la règle.

4.2. Stratégie d'application stochastique standard en MGS

Le changement de stratégie d'application des règles se fait par la spécification d'informations supplémentaires pour chaque règle (la probabilité p_i d'application de la règle R_i) ainsi que lors de l'application de la transformation à son argument.

Prenons l'exemple 2 page 255 de [BK02] : soit une urne remplie de n boules de couleur verte et deux boules de couleur jaune. Le jeu consiste a retirer de cette urne une boule tant que celle-ci n'est pas de couleur jaune. Si l'on retire toutes les balles de couleur verte, alors on perd le jeu. Le code MGS permettant d'implémenter ce jeu est immédiat :

```
Jeu[p] = {
  ‘verte  ={ P = p   }=> <undef> ;
  ‘jaune  ={ P = 1-p }=> if (count(‘verte, self) == 0)
                          then return(‘perdu)
                          else return(‘gagne)
}
```

La transformation est alors appliquée sur un état initial (un multi-ensemble composé de deux boules de couleur jaune et de cinq boules de couleur verte ici) :

```
Jeu[’fixpoint,
    ’strategy = ‘stochastic,
     p        = (let N = count(‘verte, self) in N/(N+2.0))
   ]((#5 ‘verte,  #2 ‘jaune, bag::()))
```

La déclaration `Jeu[p]` permet de déclarer `p` comme *variable locale* à la transformation ; cette variable sera remise à jour lors de chaque itération de la transformation. En effet, l'option `’fixpoint` spécifie que la transformation est itérée jusqu'à un point fixe (ici, le point fixe n'est jamais atteint, car la transformation se termine par une sortie explicite via l'usage de la commande `return()` similaire à la levée d'exception des langages fonctionnels). La variable système `self` correspond à la collection argument de la transformation ; la construction `#n ‘symbole` est un raccourci pour `n` occurences de `‘symbole`.

Cette variable `p` nous permet de spécifier pour chaque règle la valeur de la probabilité de déclenchement de celle-ci. En effet, dans cet exemple, les probabilités dépendent du temps, c'est-à-dire de l'évolution même du jeu. Lors d'un tirage d'une boule de couleur verte, la valeur `<undef>`

permet de *retirer* de la collection cette boule (matérialisée par la constante `‘verte`) en la substituant par *rien*. Le tirage d'une boule de couleur jaune, selon la présence ou non de boules de couleur verte, lève l'exception signalant un jeu gagné ou perdu.

La stratégie d'application de la transformation est stochastique par la spécification de la constante `’strategy = ‘stochastic`. Dans ce cas, chaque règle spécifie via l'option `P` la probabilité d'application de la règle. MGS vérifie que la somme des probabilités des règles soit égale à 1. Lors de chaque application de la transformation, une seule et unique règle est déclenchée, cette règle ne filtrant qu'*au plus* une seule instance de motif (application asynchrone). Si, à la règle déclenchée, ne correspond aucune instance du motif de filtrage, alors le processus de sélection d'une règle est ré-initialisé et une autre règle tirée au sort.

4.3. Stratégie stochastique spécialisée en MGS

La stratégie stochastique que l'on vient de décrire est standard dans la mesure où elle est extrêmement proche de ce que l'on trouve par exemple en Elan [BK02]. Nous proposons dans cette section une spécialisation de cette stratégie afin de prendre facilement en compte la modélisation et la simulation de systèmes biochimiques stochastiques.

La présence de phénomènes stochastiques dans un grand nombre de systèmes physiques n'est plus à démontrer. Ces dernières années, un nombre croissant de chercheurs s'est attaché à l'étude de l'aléa et du bruit présent dans les systèmes biologiques. Le lecteur intéressé pourra se reporter à [MSD04] où le détail de nombreux phénomènes stochastiques en biologie, et ce du niveau moléculaire au niveau cellulaire, est fait. Avant de présenter l'intégration de l'algorithme SSA en MGS, nous introduisons quelques notions mathématiques (dans une forme très proche de la présentation faite dans [MSD04]) nécessaire à la compréhension de l'algorithme SSA.

4.3.1. Réactions chimiques et équation maîtresse

Le comportement, au cours du temps d'une *soupe* spatialement homogène d'espèces moléculaires peut être décrit par une équation chimique dite *équation maîtresse* [Gil77] (EM), qui décrit la transition du système d'un état à un autre par des méthodes probabilistes. L'état du système est supposé décrit par la concentration de chacun des produits chimiques présents ; on peut décrire l'évolution de $p(X,t)$, la probabilité que le système soit dans l'état X à la date t, par

$$p(X, t + \Delta t) = p(X, t)[1 - \sum_{j=1}^{M}(\alpha_j \Delta t)] + \sum_{j=1}^{M} \beta_j \Delta t$$

où x_i est la quantité d'espèces chimiques i en réaction dans le système ; M est le nombre de réactions ; $\alpha_j \Delta t$ la probabilité que, le système étant dans l'état X à la date t, la j^{e} réaction ait lieu entre la date t et $t + \Delta t$; $\beta_j \Delta t$ la probabilité que la j^{e} réaction place le système dans l'état X à la date $t + \Delta t$.

En faisant tendre Δt vers 0, on obtient la forme classique de l'équation maîtresse

$$\frac{\partial p(X,t)}{\partial t}$$

On peut remarquer que la transition d'un état du système est décrite à travers le changement de la probabilité du système de se trouver dans un état donné. L'approche de l'EM consiste à essayer d'écrire un système d'équations pour chaque transition possible et de les résoudre *simultanément*. Générer une unique trajectoire est relativement facile, mais quand la dimension du problème augmente, les trajectoires possibles explosent du fait de la combinatoire et le problème devient intractable. Une solution à ce problème a été proposée par Gillespie [Gil77], qui, plutôt que d'écrire explicitement l'EM, produit les trajectoires en choisissant les *réactions* et les *temps de réaction* en accord avec les distributions de probabilités de façon à ce que la probabilité de produire une trajectoire donnée soit exactement la même que la solution de l'EM.

4.3.2. Les deux algorithmes de Gillespie

Dans [Gil77], il est proposé deux algorithmes de SSA pour résoudre l'EM décrite ci-dessus, en supposant que toutes les espèces chimiques sont réparties spatialement de façon homogène. À chaque pas de temps, le système chimique est dans un seul et unique état. L'algorithme simule l'évolution temporelle du système considéré en déterminant quand et quelle réaction doit avoir lieu.

On suppose que le système se trouve dans l'état α à la date t, qu'il est composé de M espèces chimiques qui peuvent interagir selon N réactions chimiques. On dénote les réactions R_μ, avec μ variant de 1 à N. On définit alors

- $P(\tau,\mu)$ la probabilité que la prochaine réaction R_μ ait lieu dans l'intervalle de temps infinitésimal $(t+\tau, t+\tau+d\tau)$;
- c_μ la *constante stochastique de réaction*[1] de la réaction R_μ ;
- h_μ le nombre de combinaisons moléculaires distinctes pouvant activer la réaction R_μ ;
- a_μ la fonction de propension de la réaction R_μ ;
- $a_\mu dt = h_\mu c_\mu dt$ la probabilité qu'une réaction R_μ ait lieu dans l'intervalle $(t, t+dt)$.

Gillespie a montré que

$$P(\tau,\mu)d\tau = a_\mu e^{-\tau \Sigma_j a_j} d\tau$$

que l'on peut séparer en deux distributions de probabilités indépendantes données par les deux équations suivantes

$$P(\mu) = \frac{a_\mu}{\sum_j a_j}$$
$$P(\tau)d\tau = \sum_j a_j e^{-\tau \Sigma_j a_j} d\tau$$

La première équation donne la probabilité pour la réaction R_μ d'être la prochaine réaction à avoir lieu, la seconde équation prend en compte le temps écoulé τ déterminant la date à laquelle une réaction a lieu. En utilisant un générateur de nombres aléatoires et en dérivant ces deux distributions de probabilité, on obtient l'algorithme de Gillespie dont deux versions existent : la *méthode directe* et la *méthode de première réaction*.

4.3.3. Intégration en MGS de l'algorithme de Gillespie

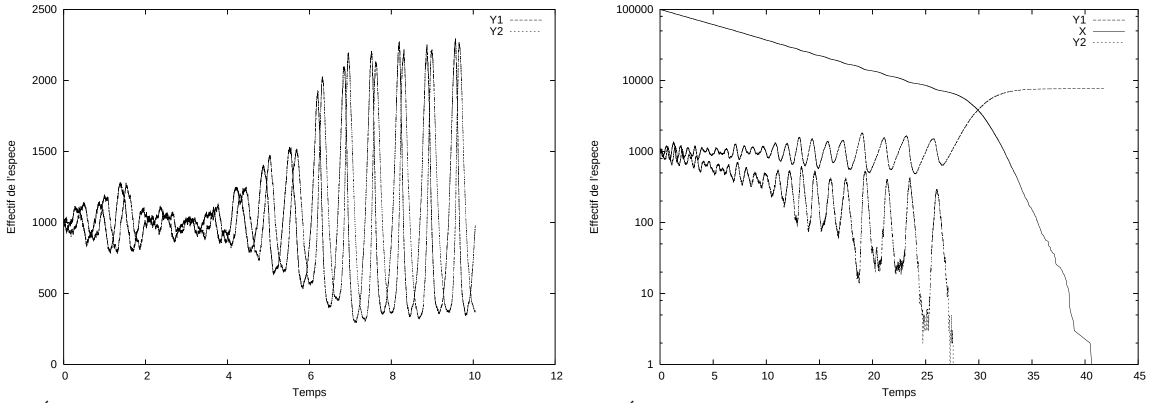
MGS intègre la *méthode de la première réaction* comme stratégie stochastique supplémentaire. Regardons son application sur l'exemple des équations autocatalytiques couplées de Lotka[2]. Les règles de réactions, accompagnées de leur *constante stochastique de réaction* spécifique, sont les suivantes :

$$\begin{array}{lll} X + Y_1 & \to X + Y_1 + Y_1 & \text{avec } C = 0.001 \\ Y_1 + Y_2 & \to Y_2 + Y_2 & \text{avec } C = 0.01 \\ Y_2 & \to Z & \text{avec } C = 10 \end{array}$$

La première règle spécifie qu'une certaine espèce de proie Y_1 se reproduit en consommant une certaine nourriture X qui ne disparaît pas lors de la consommation (X est donc une ressource supposée infinie).

[1] Déterminer la constante stochastique est une des tâches les plus difficile pour celui qui souhaite utiliser SSA pour la modélisation et la simulation de systèmes biochimiques. Le lecteur intéressé pourra consulter [DCBD03, ZDCBD03] qui décrit deux expériences dans ce domaine.

[2] Étudiées plus tard par Volterra, après la première guerre mondiale comme modèle d'étude d'un éco-système de proies-prédateurs.



(a) Évolution des espèces Y_1 et Y_2 durant 300000 itérations avec un état initial constitué de 10000 `‘X`, 1000 `‘Y1` et 1000 `‘Y2` et où la quantité de nourriture ne varie pas.

(b) Évolution des espèces Y_1, Y_2 et X, dans une variante du modèle initial où la nourriture n'est plus une ressource infinie, jusqu'à ce que la quantité de `‘X` est nulle avec un état initial constitué de 100000 `‘X`, 1000 `‘Y1` et 1000 `‘Y2` et où la nourriture consommée n'est pas réintroduite.

FIG. 3 – Résultats de la simulation sous Gnuplot des équations de Lotka pour deux jeux de paramètres différents. Les oscillations sont couplées tant que la nourriture `‘X` est présente. En absence de nourriture, les `Y1` prennent rapidement le dessus.

La seconde règle établit qu'une certaine espèce de prédateurs Y_2 se reproduit en se nourrissant des individus de l'espèce Y_1 ; enfin, la dernière règle exprime la disparition de l'espèce Y_2 par des causes naturelles.

À chaque règle, on doit attribuer une constante stochastique de réaction qui correspond à la probabilité moyenne qu'une combinaison particulière de molécules va réagir dans le prochain intervalle de temps infinitésimal *dt*. L'expression de ces équations en MGS est immédiate : les molécules sont représentées par des symboles et les constantes par les options `C` des règles :

```
trans lotka_volterra = {
 ‘X,  ‘Y1 ={ C = 0.001 }=> #2 ‘Y1, ‘X;
 ‘Y1, ‘Y2 ={ C = 0.01  }=> #2 ‘Y2;
 ‘Y2      ={ C = 10    }=> ‘Z
}
```

L'application de la transformation à une condition initiale requiert l'usage de l'option `’strategy` dont la valeur doit être `‘gillespie` :

```
lotka_volterra[’iter     = \x1.(’tau == 5.0),
               ’strategy = ‘gillespie
              ](#1000 ‘X, #100 ‘Y1, #100 ‘Y2, bag:()) ;;
```

La figure 3 montre le résultat de deux simulations du modèle.

L'état initial est un multi-ensemble car il est nécessaire de remplir l'hypothèse initiale de Gillespie, c'est-à-dire d'avoir une *soupe* chimique spatialement homogène, ce qui est vérifié dans le cas de la collection topologique de multi-ensemble [FMP00]. Chaque application de règle incrémente la variable système de MGS `’tau` par la valeur τ calculée. Lors de l'application de la transformation, il est possible d'utiliser, partout au sein de la transformation, la valeur de la variable `’tau` pour effectuer toute sorte de calculs. Les itérations ci-dessus cessent dès que `’tau` atteint ou dépasse la valeur 5. L'utilisation de `’iter` permet de disposer d'un contrôle fin sur le nombre de transformations appliquées.

Les options C dans l'expression des règles sont utilisées par MGS pour calculer la nouvelle valeur de τ après chaque application de règle. L'algorithme est le suivant :

1. pour chaque règle R_μ, suivant le motif de chemin de la règle, la fonction H_μ est calculée (par exemple, pour la première règle $H_\mu = |$`'X`$| * |$`'Y1`$|$ où $|$`'X`$|$ représente le nombre d'occurences de `'X` dans la collection),
2. pour chaque règle, la valeur $A_\mu = H_\mu * C_\mu$ est calculée (C_μ étant donnée par l'option C de la règle),
3. pour chaque règle, la valeur $\tau_\mu = \frac{1}{A_\mu} * ln(\frac{1}{random()})$ est calculée.

La règle R_i dont la valeur de τ_i est la plus faible, est choisie et *déclenchée une seule et unique fois* sur la collection argument de la transformation, après instanciation du motif de chemin. La variable système `'tau` est incrémentée par la valeur de τ_i associée à la règle.

5. Deux applications des stratégies stochastiques en MGS

Nous présentons dans les deux sections suivantes deux usages des stratégies stochastiques dans MGS, dans le domaine de l'algorithmique randomisée et dans le domaine de la modélisation de processus biologiques stochastiques.

5.1. Algorithmique randomisée

Dans [GSB94], on peut trouver un exemple d'algorithme randomisé montrant simplement comment introduire de l'aléa dans des algorithmes déterministes, et l'intérêt de ces aléas en ce qui concerne l'optimisation en moyenne. Il s'agit du problème de *sélection de chaussette*.

Pour ce problème, on considère un tiroir à vêtement contenant $2n$ chaussettes dont la moitié sont rouges et les autres bleues. La question que l'on se pose est : *combien de chaussettes doit on retirer du tiroir avant d'obtenir une paire qui correspond*.

On suppose alors qu'on ne peut garder, après les avoir tirées, que 2 chaussettes (au-delà, la solution du problème est triviale). Une solution déterministe est de tirer une première chaussette, de la garder, puis d'en prendre d'autres jusqu'à tomber sur une chaussette de la même couleur que la première. On peut traduire cette façon de faire en MGS comme suit :

```
trans choix [ première ] = {
  x => if (x == première)
       then return(‘Trouvée)
       else <undef>
} ;;

let deterministic_sockSel(première, tiroir) =
    choix[
      première  = première,
      'iter     = (1 + size(tiroir)) / 2
    ](tiroir)
```

Bien que trouvant une chaussette, le pire cas nous amène à sortir la moitié des chaussettes (c'est-à-dire n chaussettes) se trouvant dans le tiroir ; il s'avère d'ailleurs que le pire cas est en $O(n)$ pour tout algorithme déterministe. Pour améliorer nos chances de succès, il suffit de ne pas conserver en permanence la même chaussette avec soi. On utilise alors une transformation stochastique :

```
trans choix_avec_echange [ première, p ] = {
  x / (x == première) ={ P = p         }=> return(‘Trouvée) ;
  x                   ={ P = 1 - 0.5*p }=> (<undef>)
  x                   ={ P = 1 - 0.5*p }=> (première := x; <undef>)
} ;;

let stochastic_sockSel(première, tiroir) =
    choix_avec_echange[
      première  = première,
      p         = 1 / size(self),
      'iter     = (1 + size(tiroir)) / 2
      'strategy = ‘stochastic
    ](tiroir)
```

5.2. Simulation de processus biologiques complexes : le switch du phage λ

La simulation stochastique de systèmes biochimiques est intéressante quand le nombre de molécules et/ou l'intervalle de temps est faible. Nous décrivons dans cette section l'usage de MGS pour la description et la simulation d'un processus biologique très connu, la *régulation du phage lambda*.

5.2.1. Contexte biologique

Le phage lambda [Pta92] est un virus qui infecte les cellules de la bactérie *Escherichia coli*[3]. C'est un phage qui dispose de deux voies de développement :

1. réplication et *lyse* (dissolution et destruction) de la cellule hôte, libérant ainsi environ 100 virions,

2. intégration de son ADN dans l'ADN de la bactérie, et entrée en phase de *lysogénie*.

Dans la phase de lysogénie, le virus va se dupliquer de façon silencieuse à chaque fois que la bactérie se divise. De plus, une lysogénie dispose d'une immunité face à de futures infections du phage, en protégeant la bactérie de la destruction lors d'une possible nouvelle infection par un phage. Sous certaines conditions (exposition aux U.V. par exemple) une lyse peut être *induite* : l'ADN viral s'extrait de l'ADN bactérien et entame une réplication normale et une lyse.

En résumé, le virus qui infecte la bactérie a deux devenirs possibles : la *lyse* ou la *lysogénie*. Suivant le milieu, c'est l'un ou l'autre choix qui est privilégié, la décision étant sous le contrôle d'une petite région du génome du phage (une centaine de paires de bases[4], comparativement aux 48502 paires de bases de l'ADN) et de deux gènes[5] (CI et CRO[6]) et deux promoteurs[7]. On appelle cette région de régulation le *switch génétique*.

Chacune des deux protéines CI et CRO va inhiber la synthèse de l'autre et augmenter sa propre synthèse. Selon les conditions de culture, une des deux protéines va prendre le dessus sur l'autre (en terme de concentration) et on entrera alors en phase lytique ou lysogénique.

[3]Cet organisme uni-cellulaire, découvert par T. Escherich en 1885, est un hôte normal de la flore intestine chez l'homme et dont certaines souches sont toxiques.

[4]Une *paire de bases* est un couple A-T ou C-G sur la double hélice de l'ADN.

[5]Un *gène* est une région de l'ADN qui code pour une information (une protéine par exemple).

[6]CRO est l'acronyme de ***C**ontrol of **R**epressor and **O**thers*.

[7]Un *promoteur* est un site spécifique sur l'ADN en amont du gène dont le rôle est de permettre l'initiation de la transcription.

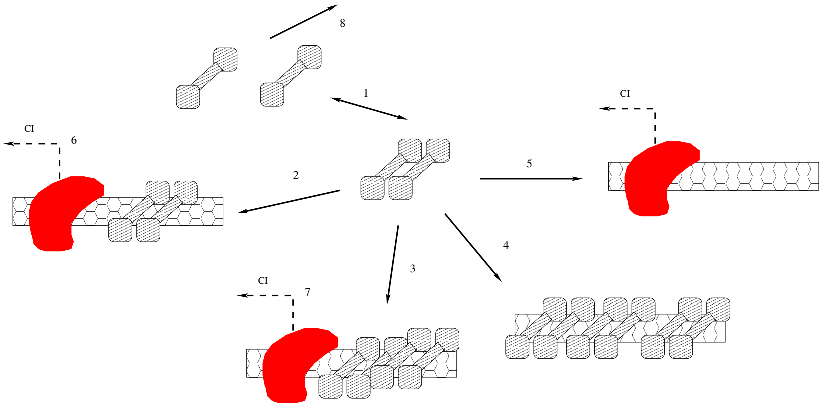


Fig. 4 – Description (provenant de [Mes05]) du réseau de régulation de la transcription du gène CI du phage λ. On y trouve l'*ARN-polymérase* en noir, un brin d'*ADN* du phage en mosaique et enfin les protéines CI qui se combinent en dimères et sont représentées en gris.

5.2.2. Les équations bio-chimiques du réseau de régulation et le programme MGS

La figure 4 représente un fragment (limité à la seule prise en compte de CI) du réseau de régulation de la transcription du phage lambda. La protéine CI en gris sur la figure joue le rôle de son propre promoteur dans le mécanisme de transcription de CI, en favorisant ou inhibant le processus. Cette protéine (équation (1)) est capable de se dimériser (association avec une autre molécule pour former un complexe) et monomériser (réaction inverse). On note respectivement `CI` et `CI2`, les états de CI sous forme de monomère et de dimère. L'équation (8) établit la dégradation de la protéine par des protéases présentes. Les équations (2)-(4) décrivent la liaison des dimères CI2 sur le brin d'ADN. Quatre états différents du brin d'ADN sont à considérer ; on les note `D`, `D1`, `D2D1` et `D3D2D1`, selon qu'aucun, un, deux ou trois dimères sont fixés sur l'ADN. L'absence (équation (5)), ou la présence d'une (équation (2)) ou deux molécules (équation (3)) de dimère favorise la transcription de CI. Par contre, la présence de trois molécules (équation (4)) de dimère inhibe sa production.

La traduction du réseau en terme d'équations chimiques (les K_i correspondent aux constantes cinétiques des réactions) est :

$$\begin{array}{llll}
CI + CI & \leftrightarrow_{K1} & CI2 & (1) \\
D + CI2 & \rightarrow_{K2} & D1 & (2) \\
D1 + CI2 & \rightarrow_{K3} & D2D1 & (3)
\end{array}
\qquad
\begin{array}{llll}
D2D1 + CI2 & \rightarrow_{K4} & D3D2D1 & (4) \\
D + P & \rightarrow_{Kt} & D + P + nCI & (5) \\
D1 + P & \rightarrow_{Kt} & D1 + P + nCI & (6) \\
D2D1 + P & \rightarrow_{a.Kt} & D2D1 + P + nCI & (7) \\
CI & \rightarrow_{K_{CI}} & & (8)
\end{array}$$

La traduction en programme MGS est immédiate. Chaque équation est traduite en une règle de réécriture et la valeur des constantes `C` est déterminée par des expériences biologiques (Cf. [KN05]). Nous donnons ci-dessous le programme MGS décrivant les lois qui gouvernent à la fois les comportements de `Cro` et de `CI` :

```
trans A = {
//
// La description de Cro
//
‘Cro          ={ C = 0.005   }=> <undef>;
#2 ‘Cro       ={ C = 0.01    }=> ‘Cro2;
‘Cro2         ={ C = 0.25    }=> #2 ‘Cro;
‘D0, ‘P       ={ C = 0.05    }=> ‘D0, ‘P, ‘Cro;
‘D0, ‘Cro2    ={ C = 9.768   }=> ‘D’3;
‘D0, ‘Cro2    ={ C = 9.768   }=> ‘D’2;
‘D’3          ={ C = 29.77   }=> ‘D0, ‘Cro2;
‘D’3, ‘P      ={ C = 0.05    }=> ‘D’3, ‘P, ‘Cro;
‘D’3, ‘Cro2   ={ C = 9.768   }=> ‘D’3D’2;
‘D’2          ={ C = 245.371 }=> ‘D’3, ‘Cro2;
‘D’2, ‘P      ={ C = 0.05    }=> ‘D’2, ‘P, ‘Cro;
‘D’3D’2, ‘P   ={ C = 0.05    }=> ‘D’3D’2, ‘P, ‘Cro;
‘D’2, ‘Cro2   ={ C = 9.768   }=> ‘D’1;
‘D’1          ={ C = 245.371 }=> ‘D’2, ‘Cro2;
//
// La description de CI
```

```
//
‘CI          ={ C = 0.0025  }=> <undef>;
#2 ‘CI       ={ C = 0.01    }=> ‘CI2;
‘CI2         ={ C = 0.25    }=> #2 ‘CI;
‘D0, ‘P      ={ C = 0.005   }=> ‘D0, ‘P, ‘CI;
‘D0, ‘CI2    ={ C = 9.768   }=> ‘D1;
‘D0, ‘CI2    ={ C = 9.768   }=> ‘D2;
‘D1          ={ C = 15.557  }=> ‘D0, ‘CI2;
‘D1, ‘P      ={ C = 0.005   }=> ‘D1, ‘P, ‘CI;
‘D1, ‘CI2    ={ C = 9.768   }=> ‘D1D2;
‘D1D2        ={ C = 15.557  }=> ‘D1, ‘CI2;
‘D1D2, ‘P    ={ C = 0.086   }=> ‘D2, ‘P, ‘CI;
‘D2          ={ C = 399.225 }=> ‘D1, ‘CI2;
‘D2, ‘P      ={ C = 0.086   }=> ‘D2, ‘P, ‘CI;
‘D2, ‘CI2    ={ C = 9.768   }=> ‘D3;
‘D3          ={ C = 2022.38 }=> ‘D2, ‘CI2
};;
```

La figure 5 montre le résultat de huit exécutions du programme MGS ci-dessus (les instructions nécessaires à la sortie sous `Gnuplot` n'ont pas été reproduites). Les trois première figures montrent le système dans un état qui n'a pas encore évolué vers une phase de lyse ou de lysogénie ; les trois figures suivantes montrent le système ayant effectué le switch dans un état de lyse (seules les molécules de `Cro` persistent) ; enfin, les deux dernières figures montrent le système ayant évolué dans un état de lysogénie (seules les molécules de `CI` persistent).

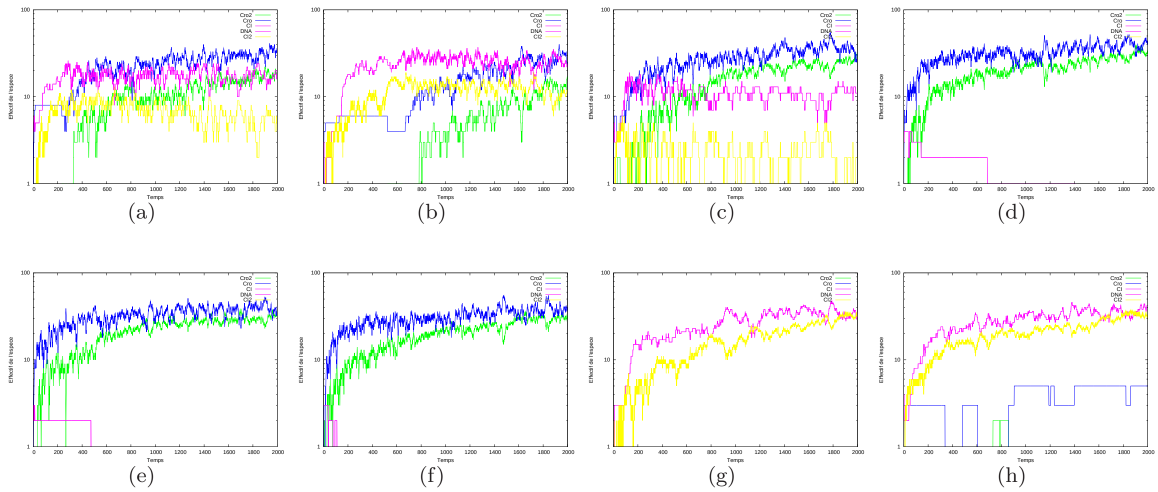


FIG. 5 – Résultats de la simulation sous Gnuplot du switch du phage λ. Les 3 premières montrent un état du système ou le switch vers la lyse ou la lysogénie n'a pas encore eu lieu ; les trois figures suivantes montrent le système en état de lyse, les deux dernières figures montrant le système en état de lysogénie. Toutes ces simulations ont consisté en 2000 itérations chacune avec un état initial constitué du multi-ensemble `(# 3 ‘CI, ‘D0, #10‘P)`. La variété des résultats montre parfaitement la nature stochastique du phénomène de switch.

6. Conclusion

Nous avons montré dans cet article le gain en expressivité amené par l'introduction d'une nouvelle stratégie d'application des transformations sur des collections topologique. Une première variante est une stratégie *standard* d'application des règles qui permet d'implémenter une grande classe d'algorithmes, les *algorithmes randomisés*. Une seconde variante, qui implémente le second algorithme SSA développé par Gillespie dans les années 70, nous permet de prendre en compte de façon extrêmement naturelle dans MGS la modélisation et la simulation de processus bio-chimiques stochastiques, montrant ainsi l'adéquation de la réécriture à la modélisation et la simulation de systèmes dynamiques. Il ne s'agit bien entendu pas de rivaliser avec les outils spécialisés (tels que `E-Cell` [Tak], `Cellware` [DMS+04] ou `SBW` [FBS+02]) mais de montrer qu'il est possible de développer rapidement en MGS des modèles réalistes, tel celui du phage lambda, et d'effectuer de courtes simulations.

Ces travaux ont ouvert la voie a de nombreuses perspectives. La première concerne évidemment la nécessaire étude de l'algorithme de filtrage et de reconstruction des topologies mis en œuvre dans MGS. En effet, si on souhaite effectuer des simulations sur des durées et des échantillons de taille raisonnable (c'est-à-dire sur des multi-ensembles de plusieurs millions d'éléments) il va être nécessaire de spécialiser l'algorithme (qui est générique et identique pour *toutes les collections topologiques* de MGS). Un premier travail à déjà été fait dans ce sens (intégration de l'écriture `#n 'x` pour exprimer le filtrage de n occurrences de la constante `'x`) mais il doit être poursuivi. De plus, il pourrait être souhaitable d'intégrer une version plus optimisée de SSA [Gil00, Gil01, GB00, PK04].

Une seconde voie, plus spéculative celle-ci, concerne l'extension des travaux initiés par Gillespie sur les *multi-ensembles*. En effet, que signifie de relâcher l'hypothèse d'homogénéité spatiale et d'appliquer SSA sur des topologies qui ne correspondent plus à des graphes complets ? On aimerait par exemple étudier des réactions bio-chimiques sur des espaces non-homogènes, structurés et compartimentés.

Remerciements

Nous souhaitons remercier les membres du groupe « Simulation et Épigénèse » de GENOPOLE, et en particulier A. Iartseva ainsi que les membres du groupe de travail autour de la modélisation du *switch du phage* λ à l'école de printemps « Modélisation de systèmes biologiques complexes dans le contexte de la génomique », pour des discussions stimulantes et pour une source d'applications en biologie. Nous sommes également reconnaissants envers J.-L. Giavitto du LaMI/UMR 8042, F. Jacquemard de l'INRIA/LSV pour leurs nombreuses questions, leur remarques constructives et leurs encouragements ainsi que F. Gaubert pour une première version de l'intégration de SSA dans MGS. Ce travail est partiellement soutenu par le CNRS, le GDR ALP et IMPG, l'Université d'Évry et le GENOPOLE-Évry.

Références

[AGG+05] Gul Agha, Michael Greenwald, Carl Gunter, Sanjeev Khanna, Jose Meseguer, Koushik Sen, and Prasanna Thati. Formal modeling and analysis of dos using probabilistic rewrite theories. In *Workshop on Foundations of Computer Security*, 2005.

[Ale82] P. Alexandroff. *Elementary concepts of topology.* Dover publications, New-York, 1982.

[BK02] Olivier Bournez and Claude Kirchner. Probabilistic rewrite strategies. applications to *ELAN*. *j-LECT-NOTES-COMP-SCI*, 2378, 2002.

[DCBD03] Xueying De Cock, Katrienand Zhang, Mónica F. Bugallo, and Petar M. Djuric. Stochastic simulation and parameter estimation of first order chemical reactions. In *12th European Signal Processing Conference (EUSIPCO-2004)*, 2003.

[DMS+04] Pawan Dhar, Tan Chee Meng, Sandeep Somani, Li Ye, Anand Sairam, Mandar Chitre, Hao Zhu, and Kishore Sakharkar. Cellware - a multi-algorithmic software for computational systems biology. *Bioinformatics*, 20(8) :1319–1321, 2004.

[ELLS01] Eiter, Lu, Lukasiewicz, and Subrahmanian. Probabilistic object bases. *ACMTDS : ACM Transactions on Database Systems*, 26, 2001.

[FBS+02] A. Finney, H. Bolouri, H. M. Sauro, J. Doyle, and M. Hucka. The Erato systems biology workbench : Enabling interaction and exchange between software tools for computational biology, September 27 2002.

[FMP00] Michael Fisher, Grant Malcolm, and Raymond Paton. Spatio-logical processes in intracellular signalling. *BioSystems*, 55 :83–92, 2000.

[GB00] Michael A. Gibson and Jehoshua Bruck. Efficient exact stochastic simulation of chemical systems with many species and many channels. *J. Chem. Physics*, 104 :1876–1889, 2000.

[GGMP02] J.-L. Giavitto, C. Godin, O. Michel, and P. Prusinkiewicz. *Modelling and Simulation of biological processes in the context of genomics*, chapter "Computational Models for Integrative and Developmental Biology". Hermes, July 2002. Also republished as an high-level course in the proceedings of the Dieppe spring school on "Modelling and simumation of biological processes in the context of genomics", 12-17 may 2003, Dieppes, France.

[Gil77] Daniel T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem.*, 81(25) :2340–2361, 1977.

[Gil00] Daniel T. Gillespie. The chemical Langevin equation. *Journal of chemical physics*, 113 :297–306, 2000.

[Gil01] Daniel T. Gillespie. Approximate accelerated stochastic simulation of chemically reacting systems. *Journal of chemical physics*, 115 :1716–1733, 2001.

[GM02a] J.-L. Giavitto and O. Michel. Data structure as topological spaces. In *Proceedings of the 3nd International Conference on Unconventional Models of Computation UMC02*, volume 2509, pages 137–150, Himeji, Japan, October 2002. Lecture Notes in Computer Science.

[GM02b] J.-L. Giavitto and O. Michel. Pattern-matching and rewriting rules for group indexed data structures. In *ACM Sigplan Workshop RULE'02*, pages 55–66, Pittsburgh, October 2002. ACM.

[GM02c] J.-L. Giavitto and O. Michel. The topological structures of membrane computing. *Fundamenta Informaticae*, 49 :107–129, 2002.

[GMC02] J.-L. Giavitto, O. Michel, and J. Cohen. Pattern-matching and rewriting rules for group indexed data structures. *ACM SIGPLAN Notices*, 37(12) :76–87, December 2002.

[GMS95] J.-L. Giavitto, O. Michel, and J.-P. Sansonnet. Group based fields. In I. Takayasu, R. H. Jr. Halstead, and C. Queinnec, editors, *Parallel Symbolic Languages and Systems (International Workshop PSLS'95)*, volume 1068 of *LNCS*, pages 209–215, Beaune (France), 2–4 October 1995. Springer-Verlag.

[Gol89] D. E. Goldberg. *Genetic Algorithm in Search, Optimization and Machine Learning*. Reading, Massachusetts, 1989.

[GSB94] Rajiv Gupta, Scott A. Smolka, and Shaji Bhaskar. On randomization in sequential and distributed algorithms. *ACM Comput. Surv.*, 26(1) :7–86, 1994.

[Hen94] M. Henle. *A combinatorial introduction to topology*. Dover publications, New-York, 1994.

[KKMA03] Sen Koushik, Nirman Kumar, José Meseguer, and Gul Agha. Probabilistic rewrite theories. Technical Report 2343, University of Illinois at Urbana Champaign, May 2003.

[KN05] Céline Kuttler and Joachim Niehren. Gene regulation in the pi calculus : simulation coperativity at the lambda switch. *Transactions on Computational Systems Biology*, 2005.

[KSMA03] Nirman Kumar, Koushik Sen, José Meseguer, and Gul Agha. A rewriting based model for probabilistic distributed object systems. In *FMOODS*, pages 32–46, 2003.

[LJ92] A. Lindenmayer and H. Jürgensen. Grammars of development : discrete-state models for growth, differentiation, and gene expression in modular organisms. In G. Ronzenberg and A. Salomaa, editors, *Lindenmayer Systems, Impacts on Theoretical Computer Science, Computer Graphics and Developmental Biology*, pages 3–21. Springer Verlag, February 1992.

[Mes05] Denis Mestivier. Modélisation déterministe de réseaux de gènes : le répresseur λ du bactériophage λ. Notes to the spring school "Modélisation de systèmes biologiques complexes dans le contexte de la génomique" on various modelization of the λ phage, Montpellier, april 2005.

[MR99] Rajeev Motwani and Prabhakar Raghavan. Randomized algorithms. In Mikhail J. Atallah, editor, *Algorithms and Theory of Computation Handbook*, pages 15–1–15–23, Boca Raton-London-New York-Washington, D.C., 1999. CRC Press.

[MSD04] Tan Chee Meng, Sandeep Somani, and Pawan Dhar. Modeling and simulation of biological systems with stochasticity. *In Silico Biology*, 4, 2004.

[Mun84] James Munkres. *Elements of Algebraic Topology*. Addison-Wesley, 1984.

[PK04] Jacek Puchalka and Andrzej Kierzek. Bridging the gap between stochastic and deterministic regimes in the kinetic simulations of the biochemical reaction networks. *Biophysical Journal*, March 2004.

[Pta92] Mark Ptashne. *A genetic switch : phage lambda and highet organisms*. 1992.

[SMG04] Antoine Spicher, Olivier Michel, and Jean-Louis Giavitto. A topological framework for the specification and the simulation of discrete dynamical systems. In *Sixth International conference on Cellular Automata for Research and Industry (ACRI'04)*, LNCS, Amsterdam, October 2004. Springer. À paraître.

[Tak] Kouichi Takahashi. E-cell, available at `http://ecell.sourceforge.net/`.

[ZDCBD03] Xueying Zhang, Katrien De Cock, Mónica F. Bugallo, and Petar M. Djuric. Stochastic simulation and parameter estimation of enzyme reaction models. In *IEEE Workshop on Statistical Signal Processing*, 2003.