Filtrage et règles de réécriture sur des structures indexées par des groupes

Julien Cohen, Olivier Michel & Jean-Louis Giavitto

LaMI U.M.R. 8042 CNRS,
Université d'Évry Val d'Essone
523, place des Terrasses de l'Agora,
91025 Évry, France.
`[jcohen,michel,giavitto]@lami.univ-evry.fr`

1. Introduction

La définition des fonctions par cas en utilisant le filtrage est l'un des atouts des langages fonctionnels actuels. Toutefois le filtrage n'est bien maîtrisé que sur des structures de données algébriques. Par exemple, on ne peut définir de fonctions par cas sur les tableaux en ML.

Nous présentons ici une nouvelle façon de définir diverses structures de données, dont les tableaux, ainsi qu'un langage de filtres permettant de définir des fonctions par cas sur ces structures. Ce nouveau style de manipulation de structures est implémenté à travers le langage de programmation `MGS`.

Les concepts présentés reposent sur des notions de topologie et unifient plusieurs modèles de calcul comme la réécriture de multi-ensembles (Gamma [BM86], le calcul chimique [BB92], les P-systèmes [Pau00]), la réécriture des chaînes (les L-systèmes [RS92], Snobol [GH91]) et les automates cellulaires ([Neu66] vus comme de la réécriture de tableaux). Les multi-ensembles et les chaînes sont des structures de données dites monoïdales [Man01, GM01b] et les techniques de réécriture associées sont maintenant bien maîtrisées. Nous nous concentrons dans cet article sur des structures non-monoïdales et plus spécialement sur une famille de structures généralisant les tableaux et proposons un style de programmation par règles de réécriture sur celles-ci.

L'article est organisé comme suit : la section suivante expose la problématique du filtrage sur les structures indexées à travers un exemple simple ; la section 3 introduit les champs de données indexés par un groupe ou *GBF* ; une interprétation géométrique de ces structures est donnée en section 4 ; une description du langage de motifs est faite en section 5 et est accompagnée d'exemples concrets en section 6 ; enfin nous proposons un algorithme original de filtrage de ces motifs sur les GBF en section 7 avant de conclure et d'esquisser la suite de nos travaux.

2. Exemples de structures indexées

Voici un exemple qui montre l'importance des structures de données indexées et l'utilité de pouvoir définir des fonctions par cas sur ces structures. Cet exemple est inspiré des simulations de gaz sur réseaux, proches des automates cellulaires. Dans ces simulations on utilise des règles de la forme $\beta \Rightarrow f(\beta)$ pour spécifier les évolutions localisées de particules réparties sur un partitionnement régulier du plan. β est un motif qui filtre une configuration planaire comme par exemple deux particules voisines. Les particules filtrées sont remplacées par $f(\beta)$. La simulation proprement dite consiste à itérer l'application des règles sur une structure de départ.

Supposons que nous voulions spécifier la rotation de particules disposées en croix sur une grille carrée (voir la figure 1). La grille est représentée par une structure du type tableau 2D et la rotation est exprimée par une règle $\beta \Rightarrow f(\beta)$ où β filtre cinq particules disposées en croix et $f(\beta)$ indique la configuration remplaçant les particules filtrées conformément à la rotation souhaitée. Notons que β ne filtre pas un sous-tableau mais une partie de la structure qui a la forme voulue.

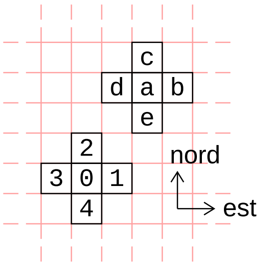


FIG. 1: *Rotation d'un motif sur une grille carrée. nord et est sont les deux générateurs.*

La fonction qui va appliquer la règle partout où cela est possible dans la grille est appelée une *transformation* et nous la noterons : `trans turn = {` $\beta \Rightarrow f(\beta)$ `;}` .

Les transformations sont des fonctions définies par cas sur des collections : les cas sont exprimés par des règles. Cette manière de définir les fonctions par cas ne correspond pas à un principe d'induction structurelle car ce n'est pas la structure de la collection qui est filtrée mais des parties arbitraires. Dans notre contexte une collection est une structure de données munie d'une topologie qui spécifie le voisinage de chaque élément dans la collection.

Pour des raisons propres aux phénomènes simulés, les physiciens préfèrent simuler les gaz sur un pavage hexagonal du plan plutôt que sur une grille carrée. La figure 2 montre une adaptation de notre exemple à ce pavage. Cette fois le motif β filtre une configuration de sept particules.

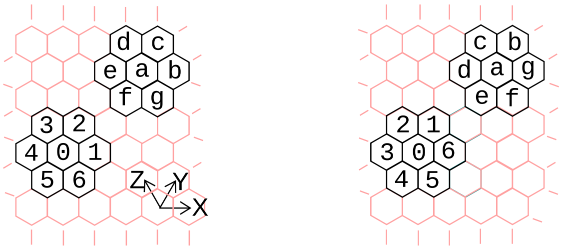


FIG. 2: *Rotation d'un motif sur un pavage hexagonal. X, Y et Z sont les générateurs.*

Pour programmer cet exemple tel qu'il est présenté on doit disposer d'un langage permettant :

1. de définir des structures de données indexées comme les tableaux,
2. d'exprimer des motifs filtrant des parties de ces structures dont la forme est arbitraire,
3. de spécifier des fonctions par cas fonctionnant par substitution des parties filtrées (ces dernières devant être disjointes entre elles).

Nous répondons à ces besoins en présentant les GBF en section 3 pour le premier point, un langage de motifs en section 5 pour le second point et les *transformation* pour le dernier point dont nous donnons des exemples en section 6.

3. Structures de données indexées par des groupes

Nous présentons rapidement dans cette section les *structures de données indexées par des groupes* ou *GBF* (*Group Based data Fields* en anglais) qui sont une généralisation de la notion de tableau [GMS96, GM01a].

Une matrice rectangulaire peut être vue comme une fonction qui à tout couple d'entiers dans $1..n \times 1..m$ associe une valeur.

On peut généraliser en considérant une structure de données représentée par une *fonction partielle à support fini* de $\mathbb{Z}^2$ dans un ensemble de valeurs. Une telle structure s'appelle un champ de données [Lis96] et permet d'avoir des tableaux à trous, des tableaux de formes variées, *etc.* Les champs de données sont une première généralisation des tableaux mais sont limités par la topologie sous-jacente à $\mathbb{Z}^n$: dans $\mathbb{Z}^n$ les voisins d'un point x sont les points $x \pm e_i$ où les e_i sont les vecteurs de la base canonique. Il n'est pas possible dans ce cadre de représenter le voisinage d'un pavage hexagonal comme celui de la figure 2. Pour ceci il faut se tourner vers les GBF.

La remarque fondatrice de la notion de GBF est que l'ensemble des indices d'un tableau n'est pas un ensemble quelconque mais est généralement muni d'opérations permettant, à partir d'un indice, de calculer un autre indice. Par exemple, selon [dLG95], 99% des accès aux tableaux dans les programmes scientifiques correspondent à une fonction linéaire. Ces opérations de manipulation d'indices permettent d'exprimer comment passer d'un élément du tableau à un élément *voisin*. Les indices sont donc vus comme des ensembles de *positions* et les opérations sur les indices comme des opérations de déplacement dans la structure de données indexée. L'idée des GBF est d'utiliser une structure de groupe mathématique pour formaliser ces déplacements (*cf.* la section 4).

Les GBF sont des structures de données représentées par une fonction partielle d'un *groupe* vers un ensemble de valeurs. L'anneau, le tore, le pavage hexagonal, la matrice 3D, l'arbre s'expriment uniformément *via* les GBF. Ce type de structure permis par les GBF est très utilisé en simulation de systèmes dynamiques.

La déclaration d'un type de GBF se fait en décrivant le groupe des indices sur lequel il se fonde *via* une présentation finie de groupe, c'est-à-dire une liste de générateurs g_i et une liste d'égalités entre combinaisons linéaires de ces générateurs. Par exemple considérons le groupe abélien $(\mathbb{Z}^2, +)$ (nous noterons $+$ l'opération de chaque groupe). La présentation `< nord,est ; nord+est=est+nord >` est une présentation finie de ce groupe. Dans `MGS` on peut déclarer le type de GBF associé à ce groupe ainsi :

```
type GBF grille = < nord,est ; nord+est=est+nord >
```

On peut omettre les équations spécifiant la commutativité des générateurs car elles sont implicites dans `MGS` lorsqu'on utilise la notation `<...>` pour la présentation du groupe. Dans le cas de groupes non-abéliens on utilisera la notation `<|...|>`. Les arbres par exemple peuvent être représentés par des GBF non-abéliens mais nous nous restreignons ici sur les GBF abéliens.

Le type de GBF hexagonal correspondant à la figure 2 est défini par

```
type GBF hexa = <X,Y,Z ; X+Z=Y>
```

(les équations de commutativité sont omises).

C'est la notion de *graphe de Cayley* qui nous permet de donner la représentation géométrique des GBF `grille` et `hexa` des figures 1 et 2. Nous présentons le lien entre un GBF et sa représentation topologique dans la section suivante.

Dans la suite du document, étant donné une présentation de groupe, nous noterons $\mathcal{G}$ le groupe en question, **G** le type de GBF associé et `g` une valeur de type **G**.

4. Représentation topologique des groupes

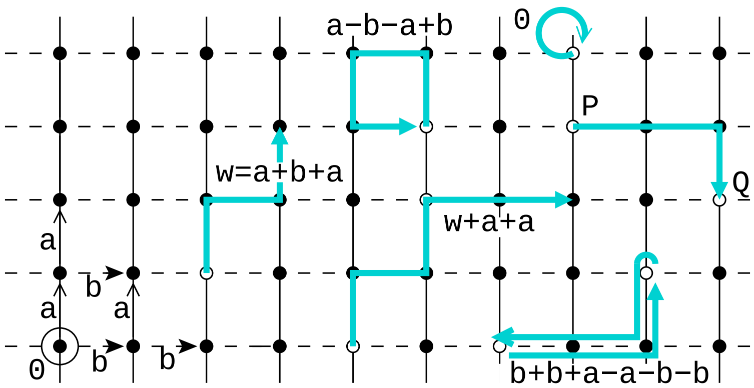


FIG. 3: *Liens entre les graphes de Cayley et la théorie des groupes.*

Étant donné une présentation finie d'un groupe $\mathcal{G}$ le graphe de Cayley associé est le graphe (E, V) tel que E est égal à l'ensemble des éléments de $\mathcal{G}$ et il existe un arc $(e1, e2)$ dans V si et seulement si il existe un générateur u de la présentation tel que $e1 + u = e2$.

Ce graphe permet une interprétation topologique :

- Les éléments de $\mathcal{G}$ sont des positions dans un espace discret.
- Une valeur $\mathtt{g}$ de type $\mathbf{G}$ peut être vue comme une fonction f_g qui associe une valeur à des positions de cet espace, pour tout couple (p, v) tel que $f_g(p) = v$ on dit que (p, v) est une valeur positionnée dont la valeur est v et dont la position est p.
- Un générateur u de la présentation exprime une *translation élémentaire* dans le graphe d'une position p à la position $p + u$, et plus généralement, tout élément du groupe peut exprimer une translation.
- On dit que deux positions p_1 et p_2 sont voisines lorsqu'il existe une translation élémentaire (*i.e.* un générateur) u tel que $p_1 + u = p_2$ (p_2 est u-voisin de p_1) ou $p_1 = p_2 + u$ (p_1 est u-voisin de p_2), cette relation de voisinage est notée par une virgule «,».
- On appelle *chemin* toute suite finie de positions. Un chemin peut être translaté par t en ajoutant t à toutes ses positions. Dans les chemins que nous manipulerons, deux positions consécutives seront voisines.
- On appelle *chemin relatif* (ou *mot*) une somme de générateurs et d'inverses de générateurs. On peut déduire un chemin à partir d'un chemin relatif et d'une position de départ.

Ces considérations permettent de donner une interprétation topologique à tout type de GBF.

La figure 4 montre le graphe de Cayley associé au type de GBF abélien défini par la présentation `< a,b >`.

Un chemin relatif est fermé s'il est égal à 0 (l'élément neutre du groupe). On distingue deux types de chemins relatifs fermés : ceux présents dans tous les graphes de Cayley consistant à avancer et revenir sur ses pas comme $b+b+a-a-b-b$ et ceux spécifiques au groupe considéré comme $a-b-a+b$

sur le schéma. Par ailleurs la composition des chemins relatifs correspond à l'addition du groupe.

Dans les figures 1 et 2 les nœuds du graphe de Cayley sont représentés par des cellules carrées et hexagonales et deux cellules sont adjacentes si et seulement si les positions correspondantes sont voisines. On peut interpréter l'équation `X+Z=Y` ainsi : une translation selon Y est équivalente à une translation selon X suivie d'une translation selon Z.

Les espaces décrits par une présentation finie sont dits *uniformes* car les positions ont toutes le même nombre de voisins.

Les GBF permettent de décrire des espaces tels que :

- des arbres n-aires par des présentations de groupes libres à n générateurs [Ser77] ;
- des grilles à n dimensions par des groupes abéliens libres à n générateurs [Mic96] ;
- des grilles circulaires ou enroulées [Mic96] ;
- des partitionnements archimédiens du plan [Cha95].

5. Un langage générique pour décrire des motifs de chemins

Les transformations sont des fonctions qui agissent sur les GBF définies par des règles de la forme $\beta \Rightarrow f(\beta)$. Une règle est composée d'un motif β et d'une expression $f(\beta)$. L'application d'une transformation à une collection `c` se fait en substituant les parties de `c` filtrées par les motifs β par l'expression en partie droite de la règle correspondante dans laquelle β est lié à la partie filtrée.

Nous présentons un langage de motifs de chemins pour filtrer des parties de GBF, en effet un chemin permet de décrire une sous-collection tout en spécifiant un ordre de parcours.

Un motif de chemin peut être un motif élémentaire comme une variable, un motif simple comme une répétition ou un motif simple lié à un motif par une direction. On peut également donner un nom à un motif. La grammaire est la suivante :

Motif	::=	*Simple* | *Simple Dir Motif* | *Motif* `as` id | `(`*Motif*`)`
Simple	::=	*Elem* | id/exp | *Elem Dir*`+` | *Elem Dir*`*`
Elem	::=	cte | id | _ | `<undef>`
Dir	::=	, | |$d_1, \ldots, d_n$`>`

où *cte* symbolise les constantes, *id* les variables, *exp* les expressions booléennes et les d_i symbolisent des combinaisons de générateurs. Les motifs s'interprètent de la manière suivante :

variable : une variable x de motif filtre une position p de la collection contenant une valeur v. x peut alors être utilisée dans une garde ou en dehors du motif pour dénoter la valeur filtrée v.

Si la variable n'apparaît pas ailleurs, on peut utiliser le motif _ afin de ne pas nommer inutilement une position filtrée.

constante : un motif c où c est une constante filtre une position dont la valeur est c.

vide : le motif `<undef>` filtre une position qui n'a pas de valeur. `<undef>` ne peut apparaître que comme voisin d'un élément bien défini.

voisinage : `m dir m'` filtre un chemin $p_0; \ldots; p_k; p_{k+1}; \ldots; p_l$ où $p_0; \ldots; p_k$ (resp. $p_{k+1}; \ldots; p_l$) est filtré par `m` (resp. `m'`) tel que p_k et p_{k+1} soient voisins selon la direction *dir*.

Ceci signifie que si `dir` est de la forme `|d1,...,dq>` alors p_k et p_{k+1} doivent être voisins selon l'une des directions `di` (chaque `di` est une combinaison de générateurs).

Par exemple `x |e> _ |n,-n> y` filtre trois positions telles que la seconde est un `e`-voisin de la première et la troisième est un `n`-voisin ou un `(-n)`-voisin de la seconde.

garde : `m/e` filtre un chemin vérifiant `e`. Par exemple `x,y/x>y` filtre deux positions p_0 et p_1 telles que la valeur en p_0 est plus grande que la valeur en p_1.

répétition : le motif `b dir*` peut filtrer un chemin vide, une position filtrée par le motif élémentaire `b` ou une suite de positions, chacune filtrée par `b` et chacune voisine de la précédente selon la direction `dir`. L'étoile est utilisée en référence aux expression régulières où elle exprime une répétition éventuellement nulle.

Si `b` est une variable `x` alors la valeur dénotée par `x` en dehors de ce motif est la valeur de la sous-collection filtrée.

Lorsque `dir` est la virgule alors on peut utiliser le raccourci syntaxique `x*`.

`x dir+` filtre les mêmes chemins que `x dir*` sauf le chemin vide.

nommage : La construction `m as x` sert lier `x` à la sous-collection filtrée par `m`. On peut écrire par exemple `1, x |n>+, 3 as Z` et `Z` dénotera en dehors du motif la sous-collection filtrée par `1, x |n>+, 3`.

Le filtrage consomme les positions. Un nom de variable ne peut être utilisé qu'une fois dans un motif excepté comme argument d'une garde (linéarité gauche au sens de [DJ90]). Ainsi le motif `x,x` n'est pas valide mais on peut écrire `x,y/y=x` pour filtrer deux éléments voisins qui ont une même valeur.

Dans la règle `x/x>0 => x+1` le motif `x/x>0` filtre un chemin composé d'une position p_0 et `x` dénote la valeur associée à la position p_0 dans la collection filtrée. Pour que cette règle s'applique il faut que `x>0`.

Dans la règle `(x, y) as Z => map f Z` si le chemin filtré est $p_0; p_1$ alors `Z` dénote un GBF égal au GBF filtré sur les positions p_0 et p_1 et non-défini sur les autres positions.

6. Exemples de programmes

Motifs simples

La figure 4 montre des motifs valides sur le type de GBF *grille*. Le motif `x,y` peut filtrer deux éléments disposés selon quatre configurations différentes. Les trois derniers motifs filtrent des parties dont la taille n'est pas connue à l'avance.

Map

La fonction *map* est un exemple trivial de transformation qui s'écrit uniformément sur toutes les collections en `MGS`. Ainsi écrite, elle peut s'appliquer à des ensembles, des multi-ensembles, des séquences et des GBF par exemple [MGC02].

```
trans map (f) = { x => f(x) ;}
```

Tout élément d'une collection est filtré par le motif `x` et donc tous les éléments de la collection sont substitués par leur image par `f`.

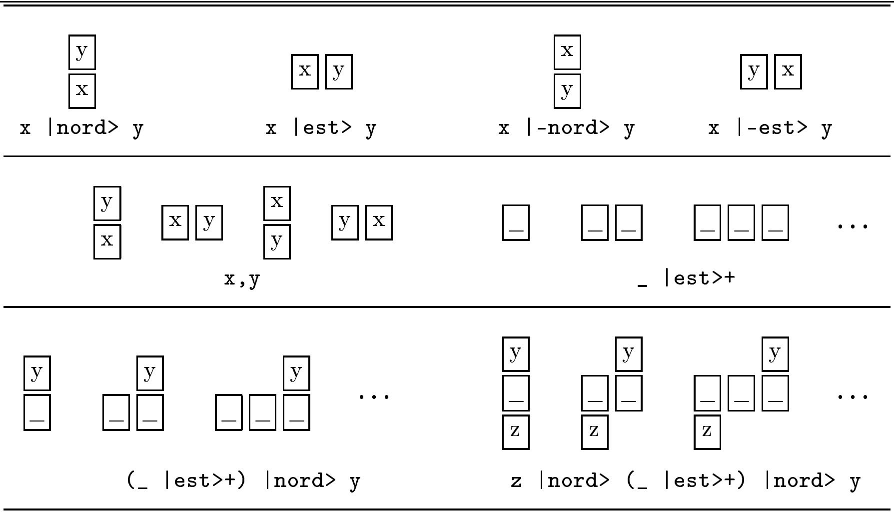


FIG. 4: *Exemples de motifs sur le GBF* grille *et parties filtrées correspondantes.*

Labyrinthe

Considérons un labyrinthe représenté par un GBF où la valeur 1 symbolise l'entrée, la valeur 2 représente les couloirs et 3 représente la sortie. Le programme suivant trouve un chemin entre l'entrée et la sortie et le marque par des 0.

```
trans laby = { 1, (2* as z), 3 => 1, (map (\x.0) z), 3 ;}
```

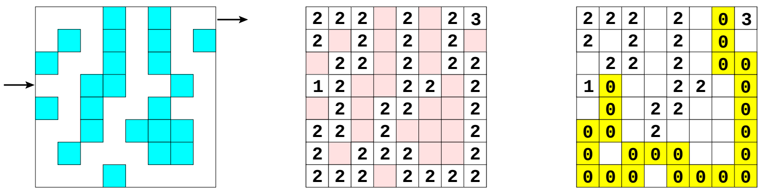


FIG. 5: *Chemins dans un labyrinthe. Le premier schéma est le labyrinthe à modéliser, le second est sa représentation dans le programme où les zones grisées ne contiennent pas de valeur. Le troisième dessin est le résultat du programme où la solution trouvée à été mise en évidence.*

Eden

Eden est un modèle simple de croissance [YPQ58] parfois utilisé pour modéliser la croissance de villes, de tumeurs ou de populations végétales. Le plan est partitionné en cellules vides ou occupées. À chaque étape de temps chaque cellule occupée dont une cellule voisine est libre se duplique dans cette dernière. La transformation correspondante s'écrit

```
trans Eden = { x, <undef> => x, x ;}
```

Une cellule pleine est représentée par une valeur dans le GBF support de la simulation et une cellule vide correspond à une position sans valeur.

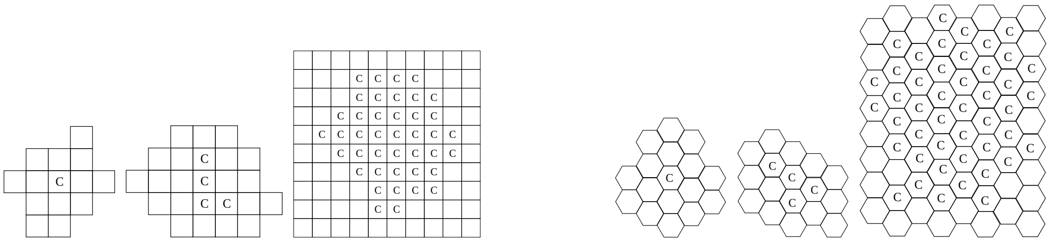


FIG. 6: *croissance Eden sur deux GBF.*

La figure 6 montre l'itération de la transformation sur deux types de GBF différents avec une seule cellule occupée au départ. La partie gauche de la figure 7 montre une variation avec trois valeurs différentes au départ, obtenue avec la même transformation.

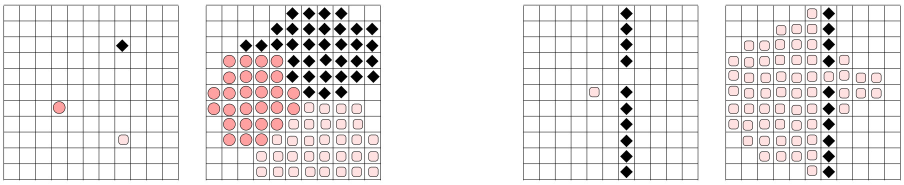


FIG. 7: *Croissance Eden avec trois espèces et avec un obstacle.*

On peut aussi définir des cellules «mortes» qui ne peuvent être occupées, il faut dans ce cas adapter la transformation :

```
trans Eden2 = {x/viv(x), <undef> => x, x;}
```

La partie droite de la figure 7 montre une croissance qui doit passer à travers un trou dans un mur de cellules mortes.

Rotation d'un motif en croix

La rotation d'un motif en croix sur un GBF de type *grille* (figure 1) s'écrit :

```
trans rota = {
      a |est> b |nord-est> c |-est-nord> d |est-nord> e => a,e,b,c,d ;
      }
```

La séquence dans la partie droite de la règle spécifie par quoi doivent être remplacés les éléments filtrés dans l'ordre où ils apparaissent dans le motif : a est remplacé par a, b par e, c par b, *etc.* Ceci correspond ici à une permutation circulaire des éléments périphériques de la partie filtrée et il en résulte une rotation globale du motif.

Voici la rotation du motif sur les GBF de type *hexa* :

```
trans rota_h = {
      a |X> b |Z> c |-X> d |-Y> e |-Z> f |X> g => a,g,b,c,d,e,f ;
      }
```

7. Un algorithme général de filtrage

Nous présentons dans cette section un algorithme simple pour le filtrage des motifs de chemins sur les GBF. Cet algorithme est inspiré de la technique de dérivation d'expressions régulières de J.A. Brzozowski [Brz64] que nous commençons par rappeler. Ensuite nous réduirons le langage de motifs au minimum nécessaire puis nous détaillerons l'algorithme sur le nouveau langage de motifs.

Dérivée d'une expression régulière

Soit R une expression régulière sur un alphabet A et L_R le langage reconnu par R. La dérivée de R par rapport à un symbole a de A est notée $\frac{\partial R}{\partial a}$ et est définie par

$$\frac{\partial R}{\partial a} = \{m \in A^* / am \in L_R\}$$

La dérivée d'une expression régulière peut aussi s'exprimer sous la forme d'une expression régulière. Par exemple[1] :

$$\frac{\partial a.(a+b)*}{\partial a} = (a+b)* \quad + \quad \emptyset$$

autrement dit am est reconnu par $a(a+b)*$ si et seulement si m est reconnu par $(a+b)*$.

L'expression R' dérivée de R peut être calculée en utilisant des règles inductives sur la structure de R. Ces règles suivent un schéma proche des règles de calcul de la dérivée d'une fonction réelle, d'où le terme de dérivée.

Un mot $m = a_1 a_2 ... a_n$ est reconnu par une expression régulière si et seulement si $a_2 ... a_n$ est reconnu par la dérivée de R par rapport à a_1. Ceci permet de ramener le problème de l'appartenance d'un mot à un langage au problème d'appartenance du mot vide ε à un autre langage.

L'annulateur $[R]$ d'une expression R est défini par

$$[R] = \begin{cases} \emptyset & \text{si } \varepsilon \notin L_R \\ \{\varepsilon\} & \text{si } \varepsilon \in L_R \end{cases}$$

et peut aussi être calculé en utilisant des règles simples sur la structure de R. L'annulateur est utilisé pour déterminer la *décomposition canonique* des mots de L_R :

$$L_R = [R] \cup \left(\bigcup_{a \in A} a \otimes \frac{\partial R}{\partial a} \right)$$

où $a \otimes L = \{am/m \in L\}$; notons que $a \otimes \emptyset$ vaut $\emptyset$ et $a \otimes \{\varepsilon\}$ vaut $\{a\}$.

Nous allons adapter ces idées à notre problématique : un motif de chemin jouera un rôle similaire à celui de l'expression régulière et le GBF un rôle similaire à celui de l'alphabet. Les différences sont les suivantes :

- Dériver une expression régulière sert à vérifier l'appartenance d'un mot à un langage. Dans notre cas nous nous préoccupons de l'existence de certains chemins dans un GBF.
- Parcourir un mot se fait simplement de gauche à droite alors qu'il n'existe pas de parcours canonique des GBF. En revanche un chemin dénote une partie d'un GBF ainsi qu'un parcours de cette partie.
- Les motifs de chemins peuvent contenir des expressions booléennes dont la valeur n'est calculée que lors du filtrage.

1. On applique la règle de dérivation d'un produit qui s'écrit $\frac{\partial(P.Q)}{\partial a} = \frac{\partial P}{\partial a}.Q + [P].\frac{\partial Q}{\partial a}$ où $[P]$ est l'annulateur de P tel que défini plus loin dans le paragraphe.

Les expressions de motifs

Afin de simplifier la présentation de l'algorithme nous utiliserons une version restreinte du langage de motifs mais presque aussi expressive que la première :

$$\begin{array}{rcl} \textit{Motif} & ::= & \textit{Simple} \mid \textit{Simple Dir Motif} \\ \textit{Simple} & ::= & \text{id/exp} \mid \textit{Dir} * \\ \textit{Dir} & ::= & \texttt{|}u_1, \ldots, u_n\texttt{>} \end{array}$$

Ce langage est presque équivalent au langage présenté dans la section 5, en effet on peut exprimer un motif *constante* par `x/x=`*cte* et un motif *variable* par `x/true`, où `x` est une variable ; La direction « `,` » se traduit par `|g1,...,gn,-g1,...,-gn>` où les `gi` sont les générateurs du GBF ; `x dir*` se code `dir*` (et il n'y a plus de liaison à un nom pour la répétition) ; `p dir+` s'écrit `p dir dir*`.

Par exemple le motif `x, (_|n>+) |e> y` peut s'écrire avec la nouvelle syntaxe `(x/true) |n,e,-n,-e> (z/true) |n> (|n>*) |e> (y/true)`.

Inclure le nommage des sous-motifs ou des répétitions à cet algorithme ainsi que le motif `<undef>` ne pose pas de véritable difficulté[2] mais alourdit beaucoup sa définition. Les parenthèses ne sont pas nécessaires et n'ont donc pas été incluses dans la grammaire.

Notations

Des accolades { } sont utilisées pour noter les ensembles en énumération ou en compréhension, $\emptyset$ est l'ensemble vide et $S - e$ est égal à l'ensemble S privé de l'élément e.

[ ] est la liste vide, @ est la concaténation de deux listes. La distribution $e \otimes S$ d'un élément e sur un ensemble S de listes est définie par $\{[e]@l / l \in S\}$.

Un environnement est une fonction d'un ensemble d'identifiants vers un ensemble de valeurs. L'augmentation d'un environnement E par un couple (i, v) est un nouvel environnement $E' = E + [i \mapsto v]$ tel que $E'(i) = v$ et $\forall j \neq i, E'(j) = E(j)$.

Dérivée d'un motif

Étant donné un GBF `g` représenté par un ensemble de couples $(\textit{position}, \textit{valeur})$, un environnement E, et un ensemble Π de positions non encore visitées, la dérivée d'un motif m par rapport à une position p s'écrit $\frac{\partial m}{\partial p}(\mathtt{g}, E, \Pi)$. Celle-ci dénote l'ensemble des chemins dans `g` commençant à la position p , filtrés par le motif m et ne passant que par des positions de Π. E est nécessaire à l'évaluation des gardes dans m. Π permet de savoir quels éléments peuvent être filtrés dans le GBF. Le résultat d'une dérivation est un ensemble de chemins *i.e.* de listes de positions.

L'ensemble des chemins filtrés par un motif m dans un GBF `g` est calculé par

$$\bigcup_{p \in dom(\mathtt{g})} \frac{\partial m}{\partial p}(\mathtt{g}, \epsilon, dom(\mathtt{g}))$$

où ϵ est l'environnement vide et où $dom(\mathtt{g})$ est l'ensemble fini des positions dans `g` qui ont une valeur définie.

Le calcul de la dérivée est défini par induction sur le motif dans la figure 8. La fonction `eval(E,g,e)` donne la valeur booléenne de l'expression e dans l'environnement E et dans le GBF `g`. La fonction `vois(`Π, dir, p`)` donne les positions voisines de la position p selon la direction dir et qui sont dans Π.

2. La gestion du motif `<undef>` est complexe, par exemple le motif `<undef>,1` doit être réécrit en `1,<undef>` sinon le filtrage du chemin ne termine pas.

$$\frac{\partial dir*}{\partial p}(\mathtt{g}, E, \emptyset) = \{[]\} \tag{1}$$

$$\frac{\partial M}{\partial p}(\mathtt{g}, E, \emptyset) = \emptyset \quad \text{avec } M \neq dir* \tag{2}$$

$$\frac{\partial id/expr}{\partial p}(\mathtt{g}, E, \Pi) = \text{if } eval(E + [id \mapsto p], \mathtt{g}, expr) \text{ then } \{[p]\} \text{ else } \emptyset \tag{3}$$

$$\frac{\partial dir*}{\partial p}(\mathtt{g}, E, \Pi) = \{[]\} \cup \frac{\partial (id/true\ dir\ dir*)}{\partial p}(\mathtt{g}, E, \Pi) \quad \text{où } id \text{ est une variable fraîche} \tag{4}$$

$$\begin{aligned}\frac{\partial id/expr\ dir\ M}{\partial p}(\mathtt{g}, E, \Pi) = {} & \text{let } E' = E + [id \mapsto p] \text{ and } \Pi' = \Pi - p \text{ in} \\ & \text{if } eval(E', \mathtt{g}, expr) \\ & \text{then } p \otimes \left(\bigcup_{p' \in vois(\Pi', dir, p)} \frac{\partial M}{\partial p'}(\mathtt{g}, E', \Pi') \right) \\ & \text{else } \emptyset\end{aligned} \tag{5}$$

$$\begin{aligned}\frac{\partial dir * dir' M}{\partial p}(\mathtt{g}, E, \Pi) = {} & \left(\bigcup_{p' \in vois(\Pi, dir', p)} \frac{\partial M}{\partial p'}(\mathtt{g}, E, \Pi) \right) \cup \frac{\partial id/true\ dir\ dir *\ \ dir'\ M}{\partial p}(\mathtt{g}, E, \Pi) \\ & \text{où } id \text{ est une variable fraîche}\end{aligned} \tag{6}$$

FIG. 8: *Calcul de la dérivée d'un motif. (On suppose dans les équations 3 à 6 que* $\Pi \neq \emptyset$*.)*

Les équations de la figure 8 s'interprètent comme suit :

1-2 : Seul le chemin vide peut être filtré dans un GBF vide.

3 : `x/e` filtre le chemin $[p]$ à partir de la position p si et seulement si `e` est vérifié dans E augmenté de la liaison `x`$\mapsto p$.

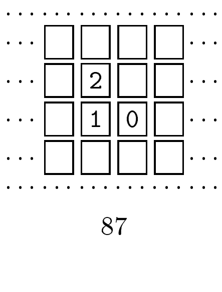
4 : Les chemins filtrés par `d*` à la position p sont le chemin vide et les chemins filtrés par `x/true d d*` à la position p (`x` est une variable fraîche).

5 : Les chemins filtrés par `(x/e) d M` en p si `e` est vérifiée sont les chemins filtrés par `M` à partir des voisins de p selon la direction `d` auxquels on a ajouté p en tête. Afin que le chemin ne s'intersecte pas, p est retiré de l'ensemble Π des positions disponibles.

6 : Les chemins filtrés par `d* d' M` sont les chemins filtrés par `M` à partir des voisins de p selon `d` (répétition nulle) et les chemins filtrés par `(x/true) d d* d' M` (répétition non nulle).

Exemples de calculs de dérivées

Pour illustrer la définition nous allons déterminer les chemins filtrant le motif `(_, 1 |nord> x)` dans le GBF suivant :



Le motif est réécrit dans la nouvelle syntaxe :

$$P = \texttt{u/true |nord,est,-nord,-est>}\ Q$$
$$Q = \texttt{(v/v=1) |nord> (x/true)}$$

Un GBF g est représenté par un ensemble de couples $(position, valeur)$, Dans le GBF *grille* chaque position peut être exprimée comme un déplacement n`|nord>`$+e$`|est>` par rapport à un point origine. Une position sera donc notée (n, e) et l'origine est choisie arbitrairement au point ayant pour valeur 1. Le GBF g et son domaine noté Π ont donc pour valeur :

$$\mathtt{g} = \{((0,0),1),\ ((0,1),0),\ ((1,0),2)\}$$

$$\Pi = dom(\mathtt{g}) = \{(0,0),\ (0,1)\ ,(1,0)\}$$

Les chemins filtrés par P son calculés par :

$$\frac{\partial P}{\partial(0,0)}(\mathtt{g},\epsilon,\Pi) \cup \frac{\partial P}{\partial(0,1)}(\mathtt{g},\epsilon,\Pi) \cup \frac{\partial P}{\partial(1,0)}(\mathtt{g},\epsilon,\Pi)$$

Calculons la dérivée à la position $(0,0)$:

$$\frac{\partial P}{\partial(0,0)}(\mathtt{g},\epsilon,\Pi') = (0,0) \otimes \bigcup_{p' \in \{(1,0),(0,1)\}} \frac{\partial Q}{\partial p'}(\mathtt{g},[u \mapsto (0,0)],\Pi')$$

où Π' vaut $\{(0,1),(1,0)\}$. Le premier terme de l'union vaut $\emptyset$ car (1,0) n'a pas de `nord`-voisin :

$$\frac{\partial Q}{\partial(1,0)}(\mathtt{g},[u \mapsto (0,0)],\Pi') = (1,0) \otimes \bigcup_{p' \in \emptyset} \frac{\partial x/true}{\partial p'}(\ldots) = (1,0) \otimes \emptyset = \emptyset$$

et le second terme de l'union vaut également $\emptyset$ car $(0,1)$ n'a pas de `nord`-voisin.

Nous avons donc $\frac{\partial P}{\partial(0,0)}(\mathtt{g},\epsilon,\Pi)$ qui vaut $(0,0) \otimes \emptyset$ donc $\emptyset$. Le même résultat est obtenu pour $\frac{\partial P}{\partial(1,0)}(\mathtt{g},\epsilon,\Pi)$. En revanche le calcul de $\frac{\partial P}{\partial(0,1)}(\mathtt{g},\epsilon,\Pi)$ donne un résultat :

$$\frac{\partial P}{\partial(0,1)}(\mathtt{g},\epsilon,\Pi) = (0,1) \otimes \bigcup_{p' \in \{(0,0)\}} \frac{\partial Q}{\partial p'}(\mathtt{g},[u \mapsto (0,1)],\Pi'')$$

où Π'' vaut $\{(0,0),(1,0)\}$ et

$$\frac{\partial Q}{\partial(0,0)}(\mathtt{g},[u \mapsto (0,1)],\Pi'') = (0,0) \otimes \frac{\partial x/true}{\partial(1,0)}(\mathtt{g},[u \mapsto (0,1), v \mapsto (0,0)],\Pi''')$$

où $\Pi''' = \Pi'' - (0,0) = \{(1,0)\}$. Or on a

$$\frac{\partial x/true}{\partial(1,0)}(\mathtt{g},\ldots,\Pi''') = \{[(1,0)]\}$$

donc le résultat final est $(0,1) \otimes ((0,0) \otimes \{[(1,0)]\})$ soit $\{[(0,1);\ (0,0);\ (1,0)]\}$, ce qui correspond au chemin attendu.

8. Conclusion

L'objectif de `MGS` est d'offrir des mécanismes permettant de construire et de faire évoluer des espaces complexes tout en les gérant fonctionnellement. La programmation par règles de transformations sur ces espaces trouve de nombreuses applications en simulation biologique (voir [GGMP02]) où des modèles mathématiques, des processus et des entités nécessitent de maintenir un état sous la forme d'un espace où des données sont associées à des lieux de cet espace. Dans ce cadre, les GBF modélisent des discrétisations régulières et uniformes d'espaces. Un sous-cas particulièrement important des GBF correspond aux tableaux d-dimensionnel (GBF abélien libre à d générateurs) et notre approche ouvre de nouvelles perspectives dans la manipulation fonctionnelle des tableaux.

Les tableaux ne se manipulent pas aussi naturellement que les listes dans les langages fonctionnels en raison de leur nature algébrique différente. Il est notamment impossible de définir des fonctions par cas sur les tableaux dans ces langages malgré les efforts de recherche dans ce sens (voir [WA85]). Le filtrage sur les tableaux avait été abordé dès [Bir77, Bak78] et plus récemment dans [Jeu92] mais la recherche de sous-tableaux plutôt que de parties de formes quelconques était un obstacle : dans une matrice rectangulaire le motif devait être rectangulaire. Notre algorithme permet le filtrage de motifs plus précis que des sous-tableaux et dans des structures moins contraintes que les tableaux.

En effet, le cadre des structures indexées par des groupes présenté ici permet de décrire uniformément des arbres et des tableaux et tire parti des caractéristiques topologiques des structures manipulées pour autoriser des motifs filtrant des sous-collections dans les GBF. L'algorithme présenté, inspiré de la reconnaissance d'expressions régulières à l'aide de leurs dérivées, sert de spécification au filtrage sur ces structures par l'interprète du langage `MGS`. Cet algorithme est également utilisé sur les ensembles, les multi-ensembles et les séquences dans la version actuelle de l'interprète `MGS`[3], ce qui montre le pouvoir unificateur de l'approche topologique (voir aussi [GM02]).

L'algorithme implanté dans l'interprète est une optimisation de celui qui est présenté. Par exemple la sélection des chemins qui seront consommés par la transformation ne nécessite pas de calculer tout l'ensemble des chemins filtrés. Nous travaillons également sur un compilateur de motifs pour un filtrage efficace basé sur la transformation de motifs à la manière de ce qui est fait dans ELAN [Mor99]. Par ailleurs l'interprète peut manipuler des motifs plus compliqués que ceux présentés comme `(x+ as X/size(X)=10)+` qui filtre une sous-collection composée de sous-collections de taille 10 qui sont connexes.

Remerciements

Nous souhaitons remercier les membres du groupe «Simulation et Épigénèse» de GENOPOLE® pour des discussions stimulantes et pour une source d'applications en biologie. Nous sommes également reconnaissants envers P. Prusinkiewicz de l'Université de Calgary et F. Delaplace de l'Université d'Évry pour leurs nombreuses questions, leur remarques constructives et leurs encouragements. Ce travail est partiellement soutenu par le CNRS, le GDR ALP et IMPG, l'Université d'Évry et GENOPOLE®-Évry.

Références

[Bak78] Theodore P. Baker. A technique for extending rapid exact-match string matching to arrays of more than one dimension. *SIAM J. Comput.*, 7(4):533–541, 1978.

[BB92] Gérard Berry and Gérard Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96:217–248, 1992.

[Bir77] Richard S. Bird. Two dimensional pattern matching. *Information Processing Letters*, 6(5):168–170, October 1977.

3. L'interprète est disponible sur la page [MGS].

[BKK+98] Peter Borovanský, Claude Kirchner, Hélène Kirchner, Pierre-Etienne Moreau, and Christophe Ringeissen. An overview of ELAN. In Claude Kirchner and Hélène Kirchner, editors, *Proceedings Second International Workshop on Rewriting Logic and its Applications, WRLA'98, Pont-à-Mousson, France, September 1–4, 1998*, volume 15 of *Electronic Notes in Theoretical Computer Science*, pages 329–344. Elsevier, 1998.

[BM86] Jean-Pierre Banâtre and Daniel Le Métayer. A new computational model and its discipline of programming. Technical Report RR-0566, Inria, 1986.

[Brz64] Janusz A. Brzozowski. Derivatives of regular expressions. *JACM*, 11(4):481–494, 1964.

[Cha95] Thomas Chaboud. About planar cayley graphs. In *Fundamentals of Computation Theory (FCT '95)*, volume 965 of *LNCS*, pages 137–142, 1995.

[DJ90] Nachum Dershowitz and Jean-Pierre Jouannaud. Rewriting systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 243–320. Elsevier Publishers, Amsterdam, 1990.

[dLG95] Damien Gautier de Lahaut and Cécile Germain. A static approach for compiling communications in parallel scientific programs. *Scientific Programming*, 4:291–305, 1995.

[GGMP02] Jean-Louis Giavitto, Christophe Godin, Olivier Michel, and Przemislaw Prusinkiewicz. *Biological Modeling in the Genomic Context*, chapter "Computational Models for Integrative and Developmental Biology". Hermes, July 2002.

[GH91] Ralph E. Griswold and David R. Hanson. String processing languages. Technical Report TR-306-91, Princeton University, Computer Science Department, February 1991.

[GM01a] Jean-Louis Giavitto and Olivier Michel. Declarative definition of group indexed data structures and approximation of their domains. In *Proceedings of the 3nd Imternational ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP-01)*. ACM Press, September 2001.

[GM01b] Jean-Louis Giavitto and Olivier Michel. `MGS`: a programming language for the transformations of topological collections. Technical Report 61-2001, LaMI – Université d'Évry Val d'Essonne, May 2001.

[GM02] Jean-Louis Giavitto and Olivier Michel. The topological structures of membrane computing. *Fundamenta Informaticae*, 49:107–129, 2002.

[GMS96] Jean-Louis Giavitto, Olivier Michel, and Jean-Paul Sansonnet. Group-based fields. In *Parallel Symbolic Languages and Systems (Int. Workshop PSLS'95)*, volume LNCS 1068, pages 209–215. Springer, 1996.

[Jeu92] Johan Jeuring. The derivation of a hierarchy of algorithms for pattern matching on arrays. In G. Hains and L. M. R. Mullin, editors, *Proceedings ATABLE-92, Second international workshop on array structures*, 1992.

[Lis96] Björn Lisper. Data parallelism and functional programming. In *Proc. ParaDigme Spring School on Data Parallelism*. Springer-Verlag, March 1996. Les Ménuires, France.

[Man01] Vincenzo Manca. Logical string rewriting. *Theoretical Computer Science*, 264(1):25–51, August 2001.

[MGC02] Olivier Michel, Jean-Louis Giavitto, and Julien Cohen. MGS : transformer des collections complexes pour la simulation en biologie. In *Journées Francophones des Langages Applicatifs*, 2002.

[MGS] The MGS Home Page : `http://mgs.lami.univ-evry.fr` .

[Mic96] Olivier Michel. *Représentations dynamiques de l'espace dans un langage déclaratif de simulation*. PhD thesis, Université de Paris-Sud, centre d'Orsay, December 1996.

[Mor99] Pierre-Etienne Moreau. *Compilation de règles de réécriture et de stratégies non-déterministes*. PhD thesis, Université Henri Poincaré - Nancy 1, 1999.

[Neu66] John Von Neumann. *Theory of Self-Reproducing Automata*. Univ. of Illinois Press, 1966.

[Pau00] Gheorghe Paun. Computing with membranes. *Journal of Computer and System Sciences*, 1(61):108–143, 2000.

[RS92] Grzegorz Rozenberg and Arto Salomaa. *Lindenmayer Systems*. Springer, Berlin, 1992.

[Ser77] Jean-Pierre Serre. *Arbres, Amalgames, SL_2*. Number 46 in Astérisque. Société Mathématique de France, 1977.

[WA85] William W. Wadge and Edward A. Ashcroft. *Lucid, the Data flow programming language*. Academic Press U. K., 1985.

[YPQ58] Hubert P. Yockey, Robert P. Platzman, and Henry Quastler, editors. *Symposium on Information Theory in Biology*. Pergamon Press, New York, London, 1958.