FIG. 1 – Une plante obtenue par réécriture de motifs

Énoncé

L'objet de ce TP [1] est d'implémenter quelques applications de la réécriture non conditionnelle de listes[2] : d'abord au flocon de neige de Von Koch puis à la modélisation de plantes. On utilisera librement les ressources de Maple sans se limiter au strict programme d'algorithmique. On autorise en particulier l'utilisation des fonctions `subs()`, `seq()`, `op()` ainsi que les types séquences et listes.
Une feuille de calcul (mw) présentant le corrigé et abordant les réécritures conditionnelles est disponible.
Le principe est de former une liste de lettres (d'abord avec « a », « l », « r »puis avec d'autres lettres encore) en partant d'une liste simple (*initiateur*) et en substituant un *motif* fixé à chaque occurence de « a ». Cette substitution sera effectuée un certain nombre de fois suivant les capacités de la machine utilisée.
Cette suite de lettres permet de former un dessin par l'intermédiaire d'une *tortue traçante* qui se déplace dans le plan. Un *état* de la tortue est un couple (point, direction) dans le plan.
Chaque lettre est un ordre donné à la tortue. Les ordres sont interprétés de gauche à droite. Par exemple « a,l,a,a »fait avancer puis tourner à gauche puis avancer deux fois. L'angle noté α et la longueur notée l sont fixés.

- la lettre « l »fait tourner sur place la tortue d'un angle α vers la gauche (dans le sens trigonométrique).
- la lettre « r »fait tourner sur place la tortue d'un angle α vers la droite.
- la lettre « a »fait avancer la tortue de une fois le vecteur direction direction et tracer le segment correspondant.

La structure de données utilisée pour représenter un état de la tortue est une liste de quatre nombres : les deux premiers sont les coordonnées de la direction, les deux derniers sont les coordonnées de la position.

[1] d'après The Algorithmic Beauty of Plants (coll The Virtual Laboratory) Springer
[2] L-system

Partie I. Flocons de Von Koch

Dans cette partie, initialement la tortue est à l'origine du repère et dirigée suivant le premier vecteur de base. L'état initial est donc $[1, 0, 0, 0]$. L'angle α est fixé à $\frac{\pi}{3}$. La variable `dessins` est initialisée par le dessin du segment réduit à l'origine :

```
> dessins := plot([[0,0],[0,0]]):
```

On prendra bien garde à toujours utiliser `:` au lieu de `;` lorsque l'on assigne à une variable ce que renvoie un appel à `plot`.

1. Écrire des procédures `a(E)` , `l(E)` , `r(E)` dont l'argument `E` désigne une liste de 4 nombres représentant l'état de la tortue.
   Les fonctions `l(E)`, `r(E)` et `a(E)` renvoient le nouvel état de la tortue après exécution de l'ordre correspondant. De plus, la fonction `a(E)` incrémente une séquence de dessins (nommé `dessins`) en ajoutant (avec `,`) le dessin du segment parcouru. La variable `dessins` est globale.
2. a. Former une *liste* de lettres de nom `L` correspondant à la figure 2 (triangle équilatéral). La position initiale est le sommet en bas à gauche et le premier segment est horizontal.
   b. Former une *séquence* de lettres de nom `motif` correspondant au dessin de la figure 3. La position initiale de la tortue étant tout à gauche.
3. Écrire une procedure `affiche(L)` dont le paramètre `L` désigne une liste de lettres et qui affiche le dessin correspondant. En reprenant `L` et `motif` trouvés en 2., vérifier que `affiche(L)` et `affiche([motif])` forment bien les dessins

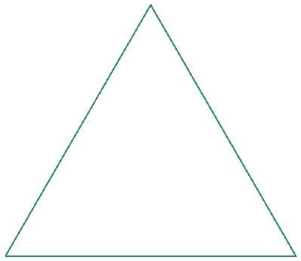

FIG. 2 – initiateur

FIG. 3 – motif

voulus. Il est utile de remarquer que si `i` désigne un entier entre 1 et `nops(L)`, alors `L[i]` est une des lettres `a`, `l`, `r` qui sont des noms de procédures. On peut donc considérer `L[i](E)`.

4. En reprenant `L` et `motif` trouvés en 2., assigner à `L` le résultat de la substitution dans `L` de `a` par motif. Recommencer quelques fois cette opération et former le dessin correspondant.

Partie II. Dessins de plantes

Pour former des dessins comme celui de la figure 1, il est nécessaire que la tortue puisse retourner (sans tracer de segment) à un de ses états précédents.
Pour cela, on va introduire deux nouvelles lettres : « e »et « m ». Ces lettres se comportent comme des parenthèses ou des crochets [3] mais il est évidemment impossible d'utiliser `(` ou `]` comme nom de procédure en maple.
On va aussi changer de structure de données. Au lieu d'un état de la tortue, on va maintenant considérer une *liste d'états*. Les ordres `a`, `l`, `r` agissent exactement comme en I. mais seulement sur le premier état de la liste, les autres étant inchangés.
Le nouvel ordre `e` agit sur la liste en « poussant »[4]. La fonction `e` pousse la liste vers la droite et insére l'état actuel de la tortue dans la premiére position. On peut imaginer que la tortue *mémorise* son état actuel. Après exécution les deux premiers éléments états de la liste sont égaux. Par exemple

$$[E_1, E_2, E_3, E_4] \to [E_1, E_1, E_2, E_3, E_4]$$

Le nouvel ordre `m` agit sur la liste en « tirant ». La fonction `m` tire la pile vers la gauche : l'état 2 devient l'état 1, l'état 3 devient l'état 2 et ainsi de suite. La tortue à la faculté de se replacer dans son premier état mémorisé. Par exemple

$$[E_1, E_2, E_3, E_4] \to [E_2, E_3, E_4]$$

1. Réécrire les procédures `a(lE)`, `l(lE)`, `r(lE)` l'argument est maintenant une *liste d'états* (nommée `lE`) de la tortue mais elles ont les mêmes fonctions que dans la première partie.
2. Écrire les procédures `e(lE)` et `m(lE)`.
3. Réécrire la procedure `affiche(L)` pour l'adapter au nouveau contexte. Vérifier avec le flocon de Von Koch.
4. Initialiser maintenant verticalement la direction de la tortue. Reproduire les modèles de développement de plantes issus de l'ouvrage cité en note (les angles sont en degré) :

initiateur	motif	angle
a	$a, e, l, a, m, a, e, r, a, m, a$	25.7
a	$a, e, l, a, m, a, e, r, a, m, e, a, m$	20
a	$a, a, r, e, r, a, l, a, l, a, m, l, e, l, a, r, a, r, a, m$	22.5

5. On obtient des modèles plus réalistes en introduisant encore une nouvelle lettre « x »représentant un *apex*. Un apex est un point de la plante où se

[3] bracketed L-systems
[4] en fait il s'agit d'une structure de pile et de fonctions usuelles pour une telle structure

forment les modifications : par exemple une nouvelle ramification. La substitution porte alors sur deux lettres. La lettre `a` est remplacée par `a,a` (la tige grandit) et la lettre `x` est remplacée par un `motifx` qui peut être compliqué. Que doit-on faire pour implémenter cette nouvelle structure ?
Reproduire les modèles suivants :

initiateur	motifx	angle
x	$a, e, l, x, m, a, e, r, x, m, l, x$	20
x	$a, e, l, x, m, e, r, x, m, a, x$	25.7
x	$a, r, e, e, x, m, l, x, m, l, a, e, l, a, x, m, r, x$	22.5

Corrigé

Partie I. Flocons de Von Koch

1. Procédures `a(E)` , `l(E)` , `r(E)`.

```
with(plots):

l:= proc(E)
  return [cos(alpha)*E[1]-sin(alpha)*E[2],
        sin(alpha)*E[1]+cos(alpha)*E[2],E[3],E[4]]:
end proc:

r:= proc(E)
  return [cos(alpha)*E[1]+sin(alpha)*E[2],
         -sin(alpha)*E[1]+cos(alpha)*E[2],E[3],E[4]]:
end proc:

a:= proc(E)
  global dessins,opt;
  local P0,P1;
  P0:= [E[3],E[4]];
  P1:= [E[3]+E[1],E[4]+E[2]];
  dessins := dessins,plot([P0,P1],opt):
  return [E[1],E[2],E[3]+E[1],E[4]+E[2]]:
end proc:
```

Noter l'appel de la bibliothèque `plots` pour pouvoir utiliser la fonction `display`.

2. a. La liste qui correspond au triangle équilatéral initiateur est `[a,l,l,a,l,l,a]`.

   b. La séquence qui correspond au motif est `a,r,a,l,l,a,r,a`.

3. La procédure d'affichage est :

```
affiche:=proc(L)
  global dessins,opt;
  local i,E;
  opt := axes=none,scaling=constrained,
         thickness=5,color="SteelBlue";
  E:=[1,0,0,0];
  dessins := plot([[0,0],[0,0]],opt):
  for i from 1 to nops(L) do
    E:= L[i](E);
  od;
  display([dessins]);
end proc:
```

Des options d'affichage ont été ajouté systématiquement.

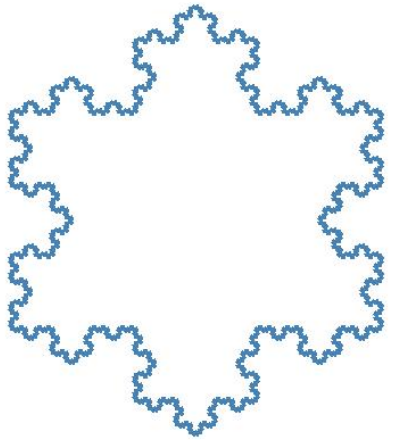

Fig. 4 – Le flocon après 4 itérations

4. Formons une boucle pour pouvoir choisir le nombre de fois où on effectue la substitution. Le dessin obtenu est en figure 4.

```
alpha:= evalf(Pi/3);
L:=[a,l,l,a,l,l,a];
motif:=a,r,a,l,l,a,r,a;
affiche(L);affiche([motif]);
nbiter :=4;
for i from 1 to nbiter do
 L:= subs({a=motif},L):
od:
affiche(L);
```

Partie II. Dessins de plantes

1. Réécritures des procédures `a(lE)` , `l(lE)` , `r(lE)`.

```
restart;
with(plots):

l:= proc(lE)
  return [[cos(alpha)*lE[1][1]-sin(alpha)*lE[1][2],
           sin(alpha)*lE[1][1]+cos(alpha)*lE[1][2],
           lE[1][3],lE[1][4]],
          seq(lE[i],i=2..nops(lE))]:
end proc:

r:= proc(lE)
  return [[cos(alpha)*lE[1][1]+sin(alpha)*lE[1][2],
          -sin(alpha)*lE[1][1]+cos(alpha)*lE[1][2],
           lE[1][3],lE[1][4]],
          seq(lE[i],i=2..nops(lE))]:
end proc:

a:= proc(lE)
  global dessins,opt;
  local P0,P1;
  P0:= [lE[1][3],lE[1][4]];
  P1:= [lE[1][3]+lE[1][1],lE[1][4]+lE[1][2]];
  dessins := dessins,plot([P0,P1],opt):
  return [[lE[1][1],lE[1][2],
           lE[1][3]+lE[1][1],lE[1][4]+lE[1][2]],
           seq(lE[i],i=2..nops(lE))]:
end proc:
```

2. Procédure `e(lE)` et `m(lE)`.

```
e:= proc(lE)
 return [lE[1],op(lE)]
end proc:

m:= proc(lE)
  return [seq(lE[i],i=2..nops(lE))]
end proc:
```

3. Réécriture de `affiche(L)`. Très peu de modifications à effectuer.

```
affiche:=proc(L)
  global dessins,opt;
  local i,lE;
  opt := axes=none,scaling=constrained,
         thickness=2,color="SeaGreen";
  lE:=[[0,1,0,0]];
  dessins := plot([[0,0],[0,0]],opt):
  for i from 1 to nops(L) do
    lE:= L[i](lE);
  od;
  display([dessins]);
end proc:
```

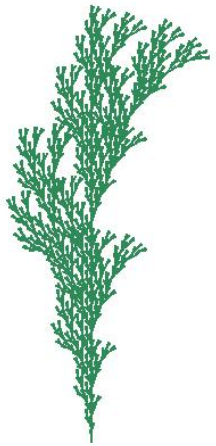

Fig. 5 – Exemple 2 pour 5 itérations

4. On procède avec les exemples comme avec le flocon. Le premier exemple donne le dessin de l'énoncé. Le deuxième est en figure 5 et le troisième en figure 6.

5. Les apex n'interviennent que pour les substitutions, il faut bien veiller à substituer à la fois pour `a` et pour `x`. En ce qui concerne `affiche`, rien n'est à changer mais des appels `x(lE)` sont effectuer. Il faut donc écrire une procédure `x(lE)` qui renvoie `lE` sans le modifier.

```
x:= proc(lE)
 return lE
end proc:
```

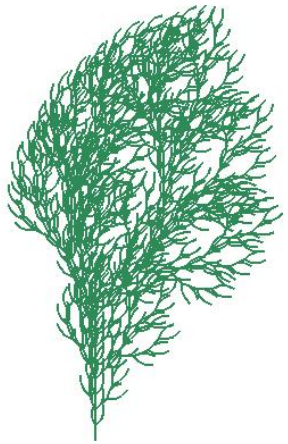

Fig. 6 – Exemple 3 pour 5 itérations

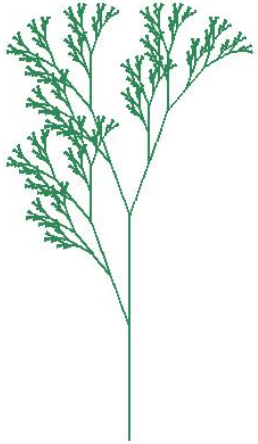

Fig. 7 – Exemple 4 pour 7 itérations

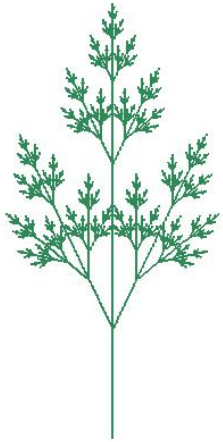

Fig. 8 – Exemple 5 pour 7 itérations

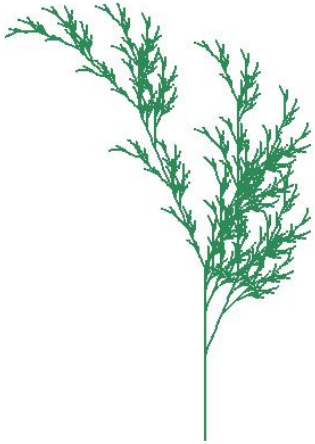

Fig. 9 – Exemple 6 pour 6 itérations