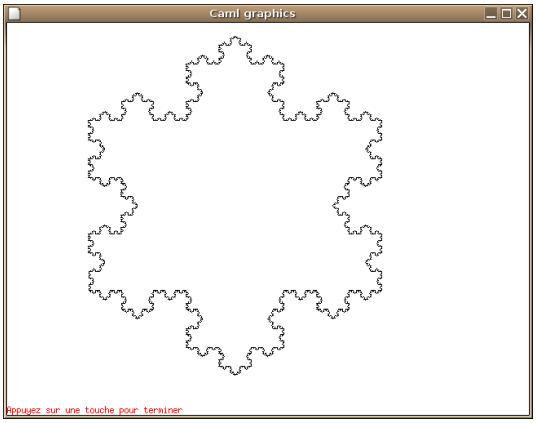
Expression Logique et Fonctionnelle . . . Évidemment

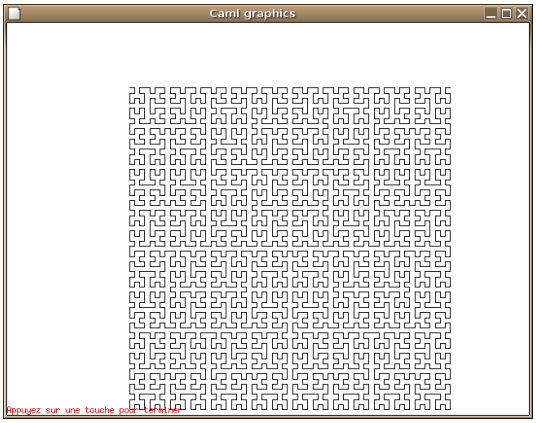
TP n° 2 : Les L-Systèmes

1 Présentation des L-systèmes

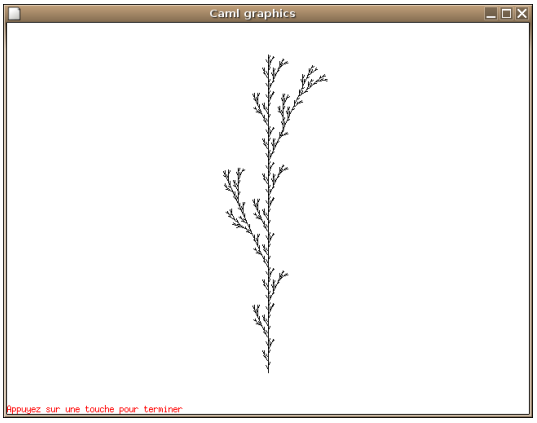
Le but de ce TP est de produire des images fractales comme celles montrées dans la figure 1.



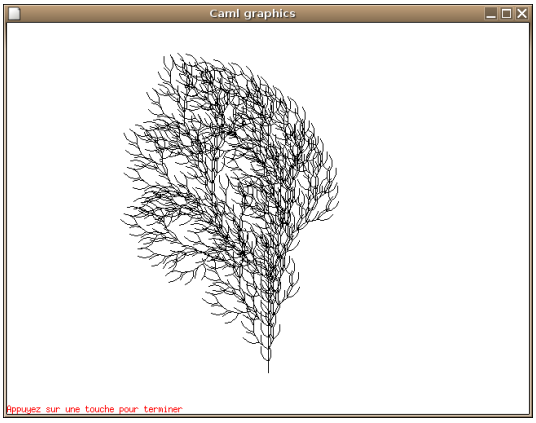
(a) flocon de von Koch



(b) courbe de Hilbert



(c) une brindille



(d) une broussaille

FIGURE 1 – Exemples d'images fractales obtenues par un L-système

Toutes ces images ont été dessinées à partir d'un *mot* les décrivant et ce mot a lui même été obtenu à l'aide d'un L-système.

1.1 Définition des L-systèmes

Les L-systèmes sont un formalisme introduit en 1968 par le biologiste Aristid Lindenmayer pour décrire la croissance des plantes. Ce formalisme décrit une grammaire permettant de dériver un mot à partir d'un mot de départ ou *axiome* et des règles de réécriture ou dérivation. Les L-systèmes que nous définissons ci-dessous sont les L-systèmes *libres* (ou *context-free*) et *déterministes*.

Définition : Un L-système est la donnée

1. d'un *alphabet* fini $A = \{a_1, a_2, \ldots, a_n\}$ de symboles ;
2. de *règles de dérivation*

$$\begin{array}{rccc} P & : & A & \to & A^* \\ & & a & \mapsto & u \end{array}$$

où A^* désigne l'ensemble des mots sur l'alphabet A ;

3. d'un *axiome* $u_0 \in A^*$.

Exemple : voici le L-système permettant d'obtenir (en partie) le flocon de von Koch ci-dessus.

1. l'alphabet $A = \{a, g, d\}$;
2. l'axiome $u_0 = a$;
3. les règles de dérivation

$$\begin{array}{rcl} a & \mapsto & agaddaga \\ g & \mapsto & g \\ d & \mapsto & d \end{array}$$

1.2 Dérivation d'un mot

Étant donné un L-système, il est possible de construire une suite $(u_n)_{n\in\mathbb{N}}$ de mots à partir de l'axiome et des règles de dérivation :

1. Le mot initial est l'axiome u_0.
2. On calcule le mot u_{n+1} à partir du mot u_n en remplaçant chacune des lettres x de u_n par son image $P(x)$ donnée par les règles de dérivation.

Exemple : avec l'exemple du L-système présenté ci-dessus, voici les premiers mots obtenus :

n	u_n
0	a
1	$agaddaga$
2	$agaddagagagaddagaddagaddagagagaddaga$

le mot u_2 est de longueur 36 et le mot suivant, u_3, est de longueur 148.

1.3 Interprétation graphique

Les mots obtenus par itération à partir de l'axiome peuvent être vus comme des programmes de commande d'une table traçante, ou encore de la célèbre tortue Logo, pour tracer des figures. Chaque symbole d'un mot possède une interprétation en terme de déplacement, tracé, orientation.

Exemple : Si on interprète le symbole a par « avancer en traçant un segment d'une certaine longueur », le symbole g par « tourner de $\frac{\pi}{3}$ radian à gauche », et le symbole d par « tourner de $\frac{\pi}{3}$ radian à droite », alors l'interprétation des trois premiers mots obtenus par le L-système de von Koch donnent les figures ci-dessous.

FIGURE 2 – Interprétation du mot $u_0 = a$

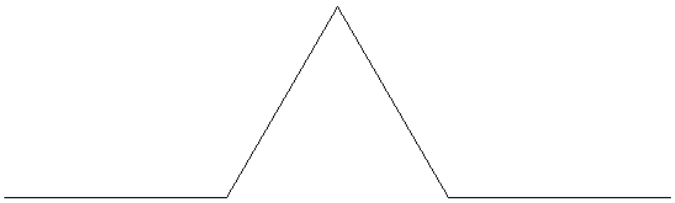


FIGURE 3 – Interprétation du mot $u_1 = agaddaga$

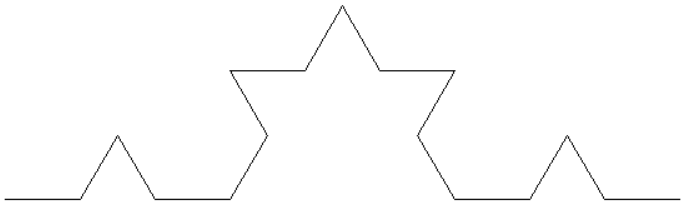


FIGURE 4 – Interprétation du mot $u_2 = agaddagagagaddagaddagaddagagagaddaga$

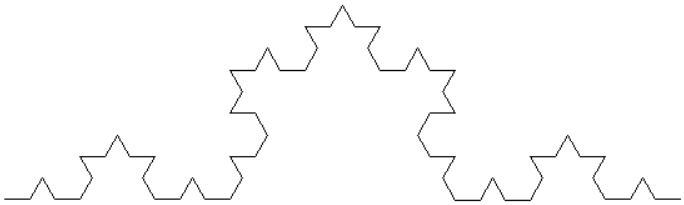


FIGURE 5 – Interprétation du mot u_3

1.4 Un exécutable

Vous pouvez récupérer ici un exécutable permettant de produire quelques images.
Pour l'utiliser il suffit de taper la commmande dans un terminal

`./l_systemes` <*ls*> <*n*>

où n est un entier positif ou nul (pas trop grand), et ls le nom d'un L-système à choisir dans la liste `vonKoch1`, `vonKoch2`, `vonKoch3`, `hilbert`, `peano`, `brindille` et `broussaille`.

1.5 Pour en savoir plus

1. Wikipedia `http://fr.wikipedia.org/wiki/L-syst%C3%A8me`.
2. *Chaos and Fractals. New Frontiers of Science*. H.-O. Peitgen, H. Jürgens & D. Saupe. Springer-Verlag. 1992.

2 Dérivation d'un mot

Toutes les déclarations de cette partie sont à mettre dans un fichier nommé `lsysteme.ml`.

2.1 Représentation des mots

Fixons (pour l'instant) l'alphabet à cinq symboles $A = \{a, g, d, x, y\}$ pour tous les L-systèmes.
Nous définissons en CAML, le type `symbole` par

```
type symbole =
  | A
  | G
  | D
  | X
  | Y
```

Cette déclaration de type énumère cinq valeurs désignées par des noms (qui doivent obligatoirement commencer par une lettre majuscule) qu'on appelle *constructeurs* en CAML.

Un mot sur l'alphabet A étant une suite finie de symboles de longueur arbitraire, les listes sont une structure de données tout à fait adaptée pour représenter un mot.

```
type mot = symbole list
```

2.2 Représentation des L-systèmes

Les L-systèmes que nous considèrerons ayant tous le même alphabet, seuls les règles de dérivation et l'axiome les distinguent. Nous les représenterons par des enregistrements à deux champs nommés `axiome` et `regles`.

```
type l_systeme = {
  axiome : mot ;
  regles : symbole -> mot
}
```

Voici quelques définitions de L-systèmes.

– Trois variantes de von Koch

```
let vonKoch1 = {
  axiome = [A];
  regles = function
    | A -> [A;G;A;D;D;A;G;A]
    | x -> [x]
}

let vonKoch2 = {
  axiome = [A;D;D;A;D;D;A;D;D];
  regles = function
    | A -> [A;G;A;D;D;A;G;A]
    | x -> [x]
}

let vonKoch3 = {
  axiome = [A;G;G;A;G;G;A;G;G];
  regles = function
    | A -> [A;G;A;D;D;A;G;A]
    | x -> [x]
}
```

– La courbe de Hilbert

```
let hilbert = {
  axiome = [X];
  regles = function
    | X -> [D;Y;A;G;X;A;X;G;A;Y;D]
    | Y -> [G;X;A;D;Y;A;Y;D;A;X;G]
    | x -> [x]
}
```

– La courbe de Peano

```
let peano = {
  axiome = [A];
  regles = function
    | A -> [A;A;G;A;G;A;G;A;A;G;A;G;A;D;A]
    | x -> [x]
}
```

– L'île de Koch

```
let ileDeKoch = {
  axiome = [A;G;A;G;A;G;A];
  regles = function
```

```
    | A -> [A;G;A;D;A;D;A;A;G;A;G;A;D;A]
    | x -> [x]
}
```

2.3 Transformation d'un mot

Question 1 Réalisez la fonction de type `l_systeme-> symbole-> mot` qui calcule le mot associé à un symbole par un L-système.
Question 2 Réalisez la fonction `transforme_mot` de type `l_systeme-> mot-> mot` qui permet de transformer un mot suivant les règles de dérivation du L-système.

2.4 Dérivation du n-ième mot

Question 3 Réalisez la fonction `derive_nieme` de type `l_systeme-> int-> mot` qui calcule le n-ième mot dérivé à partir d'un L-système. (Vous pourrez utiliser avec profit la fonction `itere` vue en cours).
Question 4 Calculez quelques mots dérivés des L-systèmes donnés en exemple. Vérifiez en particulier les longueurs annoncées dans la partie 1.2.

3 Interprétation graphique

Vous allez ici construire une procédure pour dessiner les mots produits par les L-systèmes. Pour cela, il vous faudra vous familiariser avec quelques procédures graphiques (cf 3.1) primitives, définir quelques procédures de commande de la tortue graphique (cf 3.2) et introduire quelques paramètres graphiques supplémentaires pour l'interprétation graphique des mots (cf 3.3).

Toutes les déclarations de cette partie sont à mettre dans un fichier nommé `tortue.ml`. Dans le cadre d'une utilisation avec la boucle d'interaction, ce fichier peut débuter par la commande

```
# #use "lsysteme.ml" ;;
```

3.1 Graphisme avec OCaml

Graphisme avec la boucle d'interaction : Toutes les commandes graphiques de OCaml se trouvent dans la bibliothèque `Graphics`. Cette bibliothèque ne fait pas partie des bibliothèques standard, et il convient donc de préciser au démarrage de la boucle d'interaction qu'on souhaire travailler avec elle. Pour cela, il suffit d'invoquer la boucle par la commande :

```
ledit ocaml graphics.cma
```

Quelques commandes de base : Les procédures graphiques présentées dans la table 1 doivent par défaut être préfixées par `Graphics.`. Pour se dispenser de ce préfixe, on peut utiliser la directive :

```
open Graphics ;;
```

Nom	Type	Effet
`open_graph`	`string → unit`	ouvre une fenêtre
`close_graph`	`unit → unit`	ferme la fenêtre
`clear_graph`	`unit → unit`	nettoie la fenêtre
`moveto`	`int → int → unit`	positionne le crayon
`lineto`	`int → int → unit`	trace un trait jusqu'au point
`size_x`	`unit → int`	donne la dimension horizontale de la fenêtre
`size_y`	`unit → int`	donne la dimension verticale de la fenêtre

Table 1 – Quelques procédures du module `Graphics`

- Ouverture d'une fenêtre graphique

```
# open_graph " 640x480";;
- : unit = ()
```

- dimensions horizontale et verticale de la fenêtre (en nombre de pixels)

```
# size_x ();;
- : int = 640
# size_y ();;
- : int = 480
```

– positionnement du crayon au point de coordonnées $(320, 240)$

```
# moveto 320 240;;
- : unit = ()
```

– tracé d'un segment depuis la position courante jusqu'au point de coordonnées $(160, 360)$

```
# lineto 160 360;;
- : unit = ()
```

– nettoyage de la fenêtre graphique

```
# clear_graph ();;
- : unit = ()
```

– fermeture de la fenêtre graphique

```
# close_graph ();;
- : unit = ()
```

3.2 La tortue graphique

La tortue graphique est un automate de dessin dont l'état est caractérisé par sa position courante et le cap vers lequel elle est prête à se diriger. La position courante est caractérisée par les deux coordonnées x et y de type `float` et le cap est donné par un angle par rapport à l'axe des abscisses exprimé en radians.

```
type etat = {
  x : float;
  y : float;
  cap : float;
}
```

Puisqu'il faudra utiliser les fonctions trigonométriques cos et sin, les angles sont exprimés en radians. Il est donc utile de définir la constante $\pi = 4 \arctan 1$.

```
let pi = 4.*.atan(1.)
```

La tortue peut comprendre (pour l'instant) quatre ordres :

1. *avance en traçant* : la tortue avance d'une certaine distance dans la direction de son cap en traçant un trait ; l'état qu'elle atteint est alors donné par

$$\begin{aligned} x' &= x + d \times \cos cap \\ y' &= y - d \times \sin cap \end{aligned}$$

   où d est la distance que la tortue doit parcourir ; le cap reste inchangé ;

2. *vole jusqu'à* : la tortue se rend en un point donné sans tracer ; l'état atteint par la tortue est le même que précédemment ;

3. *tourne à gauche* : le cap de la tortue change et prend la valeur

$$cap' = cap + \alpha$$

   où α est un angle donné.

4. *tourne à droite* : le cap de la tortue change et prend la valeur

$$cap' = cap - \alpha$$

Question 5 Réalisez la fonction `avance` de type `etat->float->etat` qui calcule le nouvel état de la tortue après avoir avancé d'une distance d passée en paramètre. Cette fonction aura pour effet de bord de tracer un segment sur la fenêtre graphique (supposée ouverte) de longueur d depuis la position initiale de la tortue dans la direction donnée par son cap.

Vous pourrez utiliser la fonction `int_of_float` pour convertir les coordonnées.

Vous aurez à effectuer des calculs en séquence. L'opérateur `;` est l'opérateur de séquence en CAML. Par exemple :

```
# print_string "timoleon" ;
  print_newline () ;
  10 * 25 ;;
timoleon
- : int = 250
```

est une séquence de trois instructions qui provoque l'affichage d'une chaîne de caractères, d'un passage à la ligne et de l'évaluation d'une multiplication. La valeur d'une séquence d'instructions est celle de la dernière expression (ici l'entier 250).

Question 6 Réalisez la fonction `vole` de type `etat->float->etat` qui calcule le nouvel état de la tortue après s'être déplacée sans tracer (voler) d'une distance d passée en paramètre dans la direction donnée par son cap. Cette fonction ne produit aucun tracé sur la fenêtre graphique.

Question 7 Réalisez les fonctions `tourne_gauche` et `tourne_droite` de `etat->float->etat` qui calculent le nouvel état de la tortue après avoir modifié son cap d'une valeur α passée en paramètre. Cette fonction ne produit aucun effet de bord.

Question 8 Réalisez la fonction `interprete_symbole` de type `float->float->etat->symbole->etat`. `interprete_symbole d a e s` calcule le nouvel état de la tortue selon le tableau ci-dessous

s	effet sur la tortue
A	avance d'une distance `d` en traçant
G	tourne à gauche d'un angle `a`
D	tourne à droite d'un angle `a`
autres	aucun effet

Question 9 Réalisez la fonction `interprete_mot` de type `float->float->etat->mot->unit`.

`interprete_mot d a e m` interprète les symboles du mot `m` en utilisant la distance `d`, l'angle `a` depuis l'état initial `e`.

3.3 Paramètres graphiques associés aux L-systèmes

À chaque L-système, on peut associer un certain nombre de paramètres graphiques qui le caractérisent et qui assurent que le tracé reste dans le cadre des dimensions de la fenêtre :

1. l'angle ;
2. le pas initial, longueur du segment à l'ordre 0 ;
3. le facteur de réduction de la longueur d'un segment d'un ordre à l'autre ;
4. l'état de la tortue au début du tracé.

On définit pour cela le type `param_graphiques`.

```
type param_graphiques = {
  pas : float;
  facteur : float;
  angle : float;
  etat : etat
}
```

Et voici une liste d'association des paramètres graphiques pour chaque L-système définis [1].

```
let liste_parametres = [
  (vonKoch1,{pas=540.;facteur=(1./.3.);angle=(pi/.3.);etat={x=50.;y=50.;cap=0.}});
  (vonKoch2,{pas=360.;facteur=(1./.3.);angle=(pi/.3.);etat={x=100.;y=360.;cap=0.}});
  (vonKoch3,{pas=360.;facteur=(1./.3.);angle=(pi/.3.);etat={x=100.;y=100.;cap=0.}});
  (hilbert,{pas=400.;facteur=(1./.2.);angle=(pi/.2.);etat={x=150.;y=400.;cap=0.}});
  (peano,{pas=400.;facteur=(1./.3.);angle=(pi/.2.);etat={x=100.;y=240.;cap=0.}});
  (ileDeKoch,{pas=300.;facteur=(1./.4.);angle=(pi/.2.);etat={x=150.;y=100.;cap=0.}})
]
```

1. le pas et les coordonnées initiales x et y sont adaptés à une fenêtre graphique de 640 sur 480.

Question 10 Quel est le type de cette liste ?
Question 11 Utilisez la fonction prédéfinie `List.assoc`, de type $'a -> ('a * 'b) list -> 'b$, pour déterminer le pas initial, le facteur de réduction, l'angle et l'état initial d'un L-système.
Question 12 Utiliser l'opérateur d'exponentiation `**` de type `float -> float` pour calculer la longueur des segments dans le tracé d'un mot d'ordre n d'un L-système.

3.4 La procédure dessine

Question 13 Réalisez la procédure `dessine` de type `l_systeme -> int -> unit`.
`dessine ls n pg e` dessine dans la fenêtre graphique le tracé décrit par le n-ième mot dérivé du L-système `ls` en tenant compte de ses paramètres graphiques.
Question 14 Dessinez les mots obtenus par les L-systèmes donnés dans 2.2.

4 Extensions

Vous allez maintenant voir comment réaliser les figures de brindilles ou de broussailles telles que celles montrées à la figure 1.

4.1 La mémoire

Désormais, la tortue est dotée d'une mémoire qui a la structure d'une pile. Elle est capable de mémoriser son état courant (empiler) et de rappeler le dernier état mémorisé (dépiler).

Le module `Stack` d'OCaml offre quelques fonctionnalités sur les piles dont la table 2 montre celles que vous utiliserez.

Nom	Type	Effet
`create`	`unit -> 'a Stack.t`	crée une nouvelle pile
`push`	`'a -> 'a Stack.t -> unit`	empile une donnée
`push`	`'a Stack.t -> 'a`	dépile une donnée

Table 2 – Fonctions du module `Stack`

- Création d'une variable pile

```
# let p = create ();;
val p : '_a Stack.t = <abstr>
```

- on empile 1 dans p

```
# push 1 p ;;
- : unit = ()
```

- le type de p est devenu une pile d'entiers

```
# p ;;
- : int Stack.t = <abstr>
```

- on empile successivement les entiers 2, 3 et 4

```
# push 2 p ; push 3 p ; push 4 p ;;
- : unit = ()
```

- on dépile

```
# pop p;;
- : int = 4
```

La mémoire de la tortue est donc une pile qui au départ est vide. Elle est représentée par une variable globale.

```
let memoire = create ()
```

4.2 Extension du type `symbole`

Question 15 Ajoutez les deux symboles `M` (pour mémorise), et `R` (pour rappelle) au type `symbole`.

Voici les déclarations des L-systèmes pour les brindille et broussaille.

```
let brindille = {
  axiome = [A];
  regles = function
    | A -> [A;M;G;A;R;A;M;D;A;R;A]
    | x -> [x]
}

let broussaille = {
  axiome = [A];
  regles = function
    | A -> [A;A;G;M;G;A;D;A;D;A;R;D;M;D;A;G;A;G;A;R]
    | x -> [x]
}
```

4.3 Adaptation de la procédure `interprete_symbole`

Question 16 Réalisez les fonctions

- `memorise` de type `etat->etat` qui renvoie inchangé l'état passé en paramètre après l'avoir empiler dans la mémoire ;
- et `rappelle` de type `unit->etat` qui dépile la mémoire.

Question 17 Adaptez la fonction `interprete_symbole` pour tenir compte de l'interprétation des deux nouveaux symboles.

Question 18 Ajoutez les paramètres graphiques qui suivent pour les deux nouveaux L-systèmes.

L-système	Pas	Facteur	Angle	X	Y	Cap
Brindille	390	$\frac{1}{3}$	$\frac{25\pi}{180}$ radian	320	50	$\frac{\pi}{2}$
Broussaille	280	$\frac{2}{5}$	$\frac{25\pi}{180}$ radian	320	50	$\frac{\pi}{2}$

Question 19 Dessinez des brindilles et des broussailles.