Projet de Programmation Fonctionnelle

Juliusz Chroboczek

31 octobre 2013

Introduction

Les *L-systèmes* (systèmes de Lindenmayer) sont un formalisme qui permet de générer des graphiques qui modélisent la structure des plantes, comme celui de la figure 4. Le but de ce projet est d'écrire un programme Caml qui permet à l'utilisateur de définir un L-système et de visualiser les graphiques qu'il génère.

1 L-systèmes

1.1 Chaînes parenthésées et arbres

Une *chaîne parenthésée* sur un ensemble de symboles V est une chaîne d'éléments dont des sous-chaînes peuvent être entourée de crochets « [» et «] ». Par exemple,

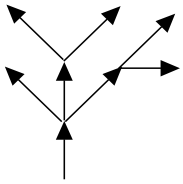
F[PF][MF[MF]F]F[PF][MF]

est une chaîne parenthésée sur l'ensemble $V = \{\mathsf{F}; \mathsf{P}; \mathsf{M}\}$.

Étant donnée une interprétation graphique des symboles, une chaîne parenthésée peut être interprétée comme représentant un arbre. Une suite de symboles (sans crochets) définit une branche sans bifurcations. Une sous-suite entre crochets définit une branche latérale ; après avoir dessiné une telle branche, on revient au point d'attachement pour dessiner la suite de l'arbre. Par exemple, si le symbole F représente un fils, le symbole P (« plus ») représente le fait de tourner vers la gauche, et le symbole M (« moins ») le fait de tourner vers la droite, alors la chaîne ci-dessus représente l'arbre de la figure 1.

1.2 Substitutions

Soit S une chaîne parenthésée, X un symbole, et T une deuxième chaîne parenthésée. Substituer X par T dans S consiste à parcourir S et à remplacer toutes les occurrences de X par T, ce qui mène à une nouvelle chaîne parenthésée, normalement plus longue que S. Par exemple, substituer F par F[PF]F dans FMF conduit à la chaîne F[PF]FMF[PF]F.



F[PF][MF[MF]F[PF][MF]

FIGURE 1 – Arbre représenté par une chaîne parenthésée

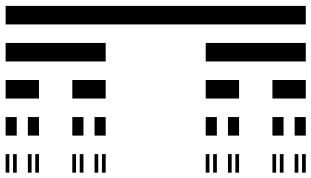


FIGURE 2 – Construction de l'ensemble de Cantor

1.3 L-systèmes

Un *L-système sans contexte* (ou simplement *L-système* dans la suite de ce document) est un triplet

$$G = (V; \, ! \, ; P)$$

où

- V est un ensemble (fini) de *symboles* ;
- ! , l'*axiome* ou *chaîne initiale*, est une chaîne parenthesée sur V ;
- P, l'ensemble des *règles de production*, est un ensemble de paires de la forme ($A \rightarrow$), où A est un symbole et une chaîne parenthésée sur V.

L'opération principale sur un L-système est l'*itération*, une opération qui pour tout entier n produit une chaîne parenthesée. Si $n = 0$, alors la n-ième itérée de G est simplement ! , l'axiome. Si $n > 0$, et si est la $(n - 1)$-ème itérée de G, alors la n-ième itérée de G est calculée en appliquant toutes les règles de production de G à selon la définition donnée au paragraphe 1.2.

1.4 Exemples

Ensemble de Cantor

$$G = (\{A; B\}; A; (A \rightarrow ABA; B \rightarrow BBB)):$$

Ce cas est dégénéré : aucune des chaînes qu'il contient ne contiennent de crochets.

0. A ;
1. ABA ;
2. ABABBBABA ;
3. ABABBBABABBBBBBBBBABABBBABA ;
4. ...

Si on interprète le symbole A comme un trait et le symbole B comme l'absence d'un trait, ce L-système implémente l'itération qui construit l'*ensemble de Cantor* (figure 2).

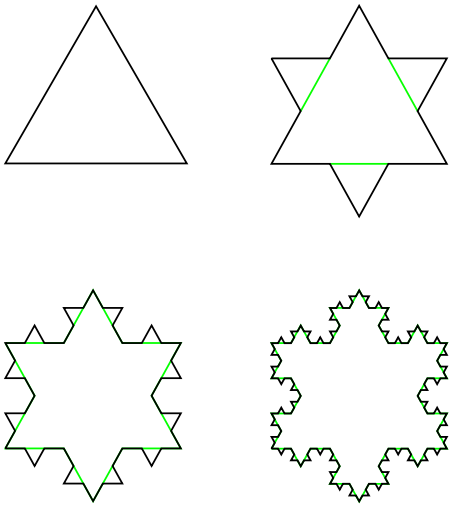


FIGURE 3 – Construction de la courbe de von Koch.

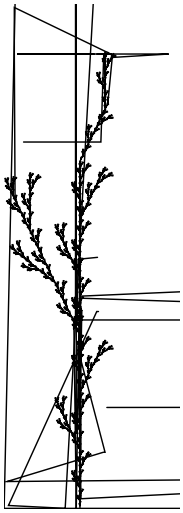


FIGURE 4 – Plante générée par un L-système (figure tirée de [1]).

Courbe de von Koch

$$G = (\{F; P; M\}; \text{FPPFPPF}; (\text{F} \to \text{FMFPPFMF})):$$

Il s'agit de nouveau d'un système ne manipulant que des chaînes ordinaires.

0. PFPPFPPF ;
1. PFMFPPFMFPPFMFPPFMFPPMFPPFMF ;
2. · · ·

Si on interprète F comme *aller tout droit*, P (« plus ») comme *tourner à gauche de 60 degrés* et M (« moins ») comme *tourner à droite de 60 degrés*, alors l'itération correspond à la construction de la courbe de von Koch (figure 3).

Plante

$$G = (\{\text{F}; \text{P}; \text{M}\}; \text{F}; (\text{F} \to \text{F[PF]F[MF]F})):$$

Cette-fois ci, il s'agit de vraies chaînes parenthésées – il y a des branches latérales qui vont s'imbriquer :

0. F ;
1. F[PF]F[MF]F ;
2. F[PF]F[MF]F[PF[PF]F[MF]F]F[PF]F[MF]F[MF[PF]F[MF]F]F[PF]F[MF]F ;
3. · · ·

Si on interprète F comme avancer, P comme tourner à gauche de 25 degrés, et M comme tourner à droite de 25 degrés, l'itération construit la « plante » de la figure 4.

2 Interprétation des arbres

Comme nous l'avons vu dans les exemples ci-dessus, l'itération d'un L-système peut être interprétée comme construisant des figures graphiques (des arbres) si l'on interprète les symboles par des opérations graphiques. Il nous faut pour cela un langage où interpréter les chaînes — celui de la tortue.

2.1 Tortue

Les *graphiques de la tortue (turtle graphics)* sont un langage de description de graphiques très simple, originellement introduit pour enseigner l'informatique aux enfants (ils sont notamment utilisés par le langage de programmation *Logo*). Intuitivement, la *tortue* est un robot qui peut avancer sans laisser de trace, avancer en laissant une trace sur le sol, ou tourner sur lui même.

Plus formellement, à tout moment l'état de la tortue consiste d'une position $(x; y)$ et d'un angle qui indique la direction vers laquelle est tournée la tortue. La tortue comprend trois commandes :

- Move(n) : avancer de n unités sans laisser de trace ;
- Line(n) : avancer de n unités en traçant une ligne ;
- Turn(n) : tourner de n degrés dans le sens trigonométrique.

Exemple La suite de commandes suivante dessine un triangle équilatéral de côté 10 :

$$\text{Line}(10) \cdot \text{Turn}(120) \cdot \text{Line}(10) \cdot \text{Turn}(120) \cdot \text{Line}(10) \cdot \text{Turn}(120):$$

2.2 Interprétation

Une interprétation est donnée par une fonction qui à tout symbole associe une suite de commandes de la tortue. Interpréter une chaîne parenthésée consiste à parcourir la chaîne en faisant faire à la tortue les actions suivantes :

- lorsqu'on rencontre un symbole S, la tortue exécute les commandes associées à S par l'interprétation ;
- lorsqu'on rencontre une chaîne entre crochets, on sauvegarde l'état $(x; y;)$ de la tortue, on interprète la chaîne contenue dans les crochets, puis on restaure l'état.

Exemple L'interprétation du deuxième exemple du paragraphe 1.4 (la courbe de von Koch) peut être codée par

$$(\text{F} \to \text{Move}(10); \text{P} \to \text{Turn}(60); \text{M} \to \text{Turn}(-60)):$$

3 Structures de données et code fourni

Les types suivants sont imposés, est il est *obligatoire* de les utiliser dans votre programme.

Symboles Le type `symbol` représente un symbole — vous pouvez utiliser un type algébrique, une chaîne, ou un caractère.

Chaînes parenthésées Le type `bracketed` représente l'arbre de syntaxe d'une chaîne parenthésée :

```
type bracketed = S of symbol
               | Seq of bracketed list
               | Branch of bracketed list;;
```

Le constructeur `Seq` représente une séquence ordinaire, tandis que le constructeur `Branch` représente une séquence entre crochets. Par exemple, si `f` et `p` sont des variables de type `symbol`, la chaîne F[P[PF]F] peut être représentée par

```
Seq [(S f);
     Branch [(S p);
             Branch [(S p); (S f)];
             (S f)]]
```

Tortue Le type `turtle_command` représente une commande de la tortue :

```
type turtle_command = Move of int
                    | Line of int
                    | Turn of int;;
```

Récriture Un système de récriture est une suite de règles de récriture, chacune représentée par une paire d'un symbole et d'une chaîne parenthésée :

```
type rewriting_system = (symbol * bracketed) list;;
```

Interprétation Enfin, une interprétation est une suite de paires d'un symbole et d'une suite de commandes de la tortue :

```
type interpretation = (symbol * turtle_command list) list;;
```

Code fourni Nous vous fournissons une sélections de L-systèmes (tirés pour la plupart de [1]) codés dans les structures de données ci-dessus dans un fichier `exemples.ml`.

4 Sujet minimal

Nous vous demandons de nous présenter un programme écrit en Caml qui permet de dessiner des figures définies par l'itération de L-systèmes. Votre programme devra être capable :

- d'itérer un L-système pendant un nombre d'itérations donné ;
- d'afficher graphiquement le résultat de l'itération selon une interprétation donnée.

Votre programme devra obligatoirement utiliser la librairie graphique fournie avec Caml.

À part les points ci-dessus, les détails d'implémentation sont laissés à votre choix. Vour pourrez par exemple nous présenter un programme qu'il faudra invoquer depuis le *toplevel* Caml (ce qui fait un peu amateur) ou nous présenter un programme utilisable depuis la ligne de commande du *shell*. Vous pourrez vous contenter d'afficher le résultat final (ce qui est un peu ennuyeux) ou vous pourrez afficher l'animation correspondant à l'itération. Vous pourrez construire une suite de commandes pour la tortue (ce qui utilisera beaucoup de mémoire) ou interpréter les chaînes parenthésées à la volée. Vous pourrez demander à l'utilisateur l'échelle à laquelle se fait le dessin (ce qui est peu pratique) ou calculer automatiquement les bornes du graphique pour choisir une échelle raisonnable. L'ensemble des L-systèmes et interprétations pourra être câblé en dur dans le programme (ce qui est peu flexible), ou vous pourrez écrire un analyseur qui permet à l'utilisateur de rentrer un L-système et une interprétation arbitraires.

Par contre, nous vous déconseillons de passer trop de temps à construire une interface graphique avec boutons, menus et boîtes de dialogue — ce n'est pas le sujet du projet, et les examinateurs aiment bien la ligne de commande.

5 Extensions

De nombreuses extensions sont possibles, et toute extension présentée sera évaluée avec intérêt et indulgence. Si vous êtes à court d'idées, voici quelques suggestions.

Analyse syntaxique Il est bien-sûr désirable de permettre à l'utilisateur d'entrer de nouveaux L-systèmes et les interprétations associées. Nous vous encourageons donc à concevoir un langage de définition de L-systèmes pratique à utiliser, et d'implémenter un analyseur syntaxique (un *parseur*) qui sera capable de lire des L-systèmes et des interprétations définies dans votre langage.

Sauvegarde Votre programme pourra être capable d'enregistrer sur disque une figure représentant le résultat de l'itération, par exemple sous format *EPS* ou *HP-GL* (facile), ou alors *PDF* ou *SVG* (un peu plus pénible). Si vous décidez d'utiliser une bibliothèque toute faite pour générer les figures, assurez-vous d'avoir compris la différence entre une image vectorielle et une image *raster*.

Couleurs Étendez votre programme pour qu'il génère des figures en couleurs. Pour cela, il faudra étendre le langage de la tortue avec des commandes permettant de changer la couleur. Comment faire pour générer des dégradés ?

Graphiques en trois dimensions Une tortue qui vole. Une projection orthonormale devrait être facile à implémenter vous-mêmes, vous n'avez probablement pas envie d'essayer de comprendre OpenGL.

6 Soumissions

Le projet sera à traiter en groupes de 2 personnes au plus (les projets soumis par des groupes de 3 personnes ou plus ne seront pas acceptés). Votre solution devra consister d'un programme écrit en Caml et utilisable sous Linux et consistera de :

- un fichier texte nommé README contenant vos noms et indiquant brièvement comment compiler et utiliser votre programme ;
- un rapport sous format PDF de 2 à 12 pages environ décrivant sommairement votre soumission, décrivant les extensions traitées, et expliquant (ou justifiant) les choix de conception ou d'implémentation que nous pourrions ne pas comprendre du premier coup ;
- tout autre fichier nécessaire à la compilation et à l'exécution, par example des fichiers d'exemples, un fichier `Makefile`, un script permettant de compiler vos sources, etc.

Votre soumission devra consister d'une seule archive compressée `tar.gz`. L'archive devra *obligatoirement* s'appeler *nom1*`-`*nom2*`.tar.gz`, et s'extraire dans un répertoire *nom1*`-`*nom2*`/`, où *nom1* et *nom2* sont les noms des deux personnes constituant le groupe. Par exemple, si vous vous appelez Pierre Corneille et Jean Racine, votre archive devra s'appeler `corneille-racine.tar.gz` et s'extraire dans un répertoire `corneille-racine/`.

Références

[1] Przemysław Prusinkiewicz and Aristid Lindenmayer. *The Algorithmic Beauty of Plants.* Springer-Verlag. 2004.