INFO0054 - Programmation fonctionnelle

Projet: Génération automatique de figures

Jean-Michel Begon

09 mars 2021

1 Aperçu

Le but de ce projet est de générer automatiquement des figures. Pour ce faire, le projet est divisé en trois étapes. La première consiste à générer de manière structurée des instructions permettant de dessiner les figures. La seconde consiste à créer le dessin en mémoire en suivant ces instructions. Finalement, la dernière étape consiste à sauver ces images au format SVG.

La génération des instructions se fera à l'aide d'un L-system (section 2). Les instructions seront exprimées dans le "langage tortue" (section 3). La section 4 décrit les L-systems correspondant à plusieurs figures.

2 L-system

2.1 Description

Formellement, un système de Lindenmayer, aussi appelé L-system, est défini par

- Un ensemble fini N de *symboles non-terminaux*. Chacun de ces symboles peut éventuellement être accompagné de paramètres.
- Un ensemble fini Σ de *symboles terminaux*. Chacun de ces symboles peut éventuellement être accompagné de paramètres.
- Un axiome qui est une séquence finie non vide de symboles de $N \cup \Sigma$.
- Un ensemble de règles de production. Chacune de ces règles est de la forme $s \to S$ où s est un symbole de $N \cup \Sigma$ et S une séquence finie de symboles de $N \cup \Sigma$. Lorsque plusieurs règles se rapportent à un même symbole s, une distribution de probabilités leur est associée.
- Un ensemble de règles de terminaison. Chacune de ces règles est de la forme $t \to T$ où t est un symbole de Σ et T une séquence finie de symboles de N. Lorsque plusieurs règles se rapportent à un même symbole t, une distribution de probabilités leur est associée.

2.2 Génération

Un L-system génère une chaîne d'ordre K de la manière suivante :

```
generate(K)
    terminate(develop(K))

terminate(string)
    for each non-terminal symbol of string
        replace it with a terminal symbol obtained by applying
        a corresponding termination rule
```

```
develop(K)
    if K == 0
        return axiom
    string = develop(K-1)
    for each non-terminal symbol of string
        replace it by applying a corresponding production rule
    return string
```

2.3 Exemple

Soit le L-system suivant :

- $N = \{A[x], B\}$
- $\Sigma = \{F, G\}$
- Axiome : $A[16]$
- Règles de production
  $A \rightarrow A[0.5x]\ B$
  $B \rightarrow G\ B$
- Règles de terminaison
  $A[x] \rightarrow F[x]$
  $B \rightarrow G$

Une chaîne non-terminale d'ordre 4 serait :

0. $A[16]$
1. $A[8]\ B$
2. $A[4]\ B\ G\ B$
3. $A[2]\ B\ G\ B\ G\ G\ B$
4. $A[1]\ B\ G\ B\ G\ G\ B\ G\ G\ G\ B$

La chaîne terminale correspondante est donc :

$$F[1]\ G\ G\ G\ G\ G\ G\ G\ G\ G\ G$$

On représentera la chaîne terminale par une liste de `string`s. La chaîne précédente sera donc :

```
(list "F[1]" "G" "G" "G" "G" "G" "G" "G" "G" "G" "G")
```

3 Interprétation graphique

3.1 Le langage tortue

Il est possible de dessiner à l'aide des chaînes terminales générées par un L-system en donnant une interprétation aux symboles : imaginons une tortue dans le plan qui effectue l'action associée à chacun des symboles suivants :

`T`	Avance d'une unité dans la direction courante en traçant une ligne (ajoutée à la polyligne courante).
`T[x]`	Avance de x unités dans la direction courante en traçant une ligne (ajoutée à la polyligne courante).
`F`	Avance d'une unité dans la direction courante sans tracer de ligne et commence une nouvelle polyligne.
`F[x]`	Avance de x unités dans la direction courante sans tracer de ligne et commence une nouvelle polyligne.
`<`	*Push* la position et la direction courante.
`>`	*Pop* la dernière position et direction sauvegardée s'il y en a une et commence une nouvelle polyligne.
`+`	Pivote d'α radians dans le sens trigonométrique.
`+[x]`	Pivote d'αx radians dans le sens trigonométrique.
`-`	Pivote d'α radians dans le sens anti-trigonométrique.
`-[x]`	Pivote d'αx radians dans le sens anti-trigonométrique.

Remarques :

— Une polyligne, aussi appelée lignée brisée ou ligne polygonale, est une suite de N segments $S_i = (P_i^s; P_i^e)$ $(i = 1, \ldots, N)$, où P_i^s et P_i^e sont respectivement les points de début et de fin du segment, vérifiant $P_i^e = P_{i+1}^s$ pour $1 \leq i < N$. En mots, il s'agit d'une suite de segments dont la fin du $i$ème coïncide avec le début du $(i+1)$ème.

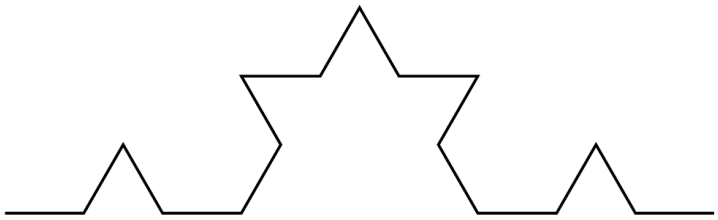
— L'angle de rotation α est un paramètre à fournir.

3.2 Exemple

Soient un angle de rotation $\alpha = 60$ degrés et la chaîne d'instruction :

`T + T - - T + T + T + T - - T + T - - T + T - - T + T + T + T - - T + T`

Le dessin résultant serait :



3.3 Représentation en mémoire du dessin

La représentation en mémoire est décrite dans le fichier `drawing.scm`. Puisque les objets sont immuables en programmation déclarative, l'état correspondant au dessin doit être re-créé après chaque instruction (`Store-passing style`). Ce mécanisme est explicité dans le fichier `turtle.scm` au travers des `turtle f-store`.

4 Figures

Bien que simple, ce langage permet de dessiner un bon nombre de figures. Dans le cadre du projet, nous allons implémenter les L-systems suivants.

Triangle de Sierpinski

- $N = \{A, B\}$
- $\Sigma = \{T, +, -\}$
- Axiome : $A\ -\ B\ -\ B$
- Règles de production
  $A \to A\ -\ B\ +\ A\ +\ B\ -\ A$
  $B \to B\ B$
- Règles de terminaison
  $A \to T$
  $B \to T$
- Angle : 120 degrés

Tapis de Sierpinski

- $N = \emptyset$
- $\Sigma = \{T, +, -, <, >\}$
- Axiome : $T\ -\ T\ -\ T\ -\ T$
- Règles de production
  $T \to T\ <\ -\ T\ -\ T\ >\ T\ <\ -\ T\ -\ T\ -\ T\ >\ T$
- Règles de terminaison : /
- Angle : 90 degrés

Dragon curve

- $N = \{D, E\}$
- $\Sigma = \{T, +, -\}$
- Axiome : D
- Règles de production
  $D \to -\ D\ +\ +\ E$
  $E \to D\ -\ -\ E\ +$
- Règles de terminaison
  $D \to -\ -\ T\ +\ +\ T$
  $E \to T\ -\ -\ T\ +\ +$
- Angle : 45 degrés

Arbre

- $N = \emptyset$
- $\Sigma = \{T, +, -, <, >\}$
- Axiome : T
- Règles de production
  $T \to T\ <\ +\ T\ >\ T\ <\ -\ T\ >\ T$ (prob. 0.33)
  $T \to T\ <\ +\ T\ >\ T$ (prob. 0.33)
  $T \to T\ <\ -\ T\ >\ T$ (prob. 0.34)
- Règles de terminaison : /
- Angle : 25.7 degrés

Plante

— $N = \{B[x]\}$

— $\Sigma = \{T[x], +[x], -[x], <, >\}$

— Axiome : $B[1]$

— Règles de production

$B[x] \rightarrow T[x] < +[5]\ B[0.5x] > < -[7]\ B[0.5x] >$
$- [1]\ T[x] < +[4]\ B[0.5x] > < -[7]\ B[0.5x] >$
$- [1]\ T[x] < +[3]\ B[0.5x] > < -[5]\ B[0.5x] >$
$- [1]\ T[x]\ B[0.5x]$

— Règles de terminaison : $B[x] \rightarrow T[x]$

— Angle : 8 degrés

Courbe de Gosper

— $N = \{A, B\}$

— $\Sigma = \{T, +, -\}$

— Axiome : A

— Règles de production

$A \rightarrow A - B - - B + A + + A\ A + B -$
$B \rightarrow + A - B\ B - - B - A + + A + B$

— Règles de terminaison

$A \rightarrow T$
$B \rightarrow T$

— Angle : 60 degrés

Un exemple de figures générées pour chacun de ces L-systems est disponible sur eCampus.

5 SVG

Afin de visualiser les figures, on les sauvegardera au format SVG (Scalable Vector Graphics). Il s'agit d'un format vectoriel d'image du W3C.

La balise `polyline` est propice à notre usage mais n'est pas obligatoire. Afin que les figures soient visibles, on veillera à ce qu'elles utilisent au mieux toute la place offerte. Une possibilité est de laisser l'interpréteur gérer cela en adaptant le `viewport` et la `viewBox` [1].

6 Implémentation

Cinq fichiers sont fournis. Le fichier `drawing.scm` défini une interface pour manipuler un dessin. Quant à `app.scm`, il permet d'orchestrer les différentes parties. Les trois autres fichiers, décrits ci-après, sont à compléter.

6.1 `lsystem.scm`

Il s'agit d'implémenter les fonctions `lsystem.generate-string`, `tlsyst.angle` et `tlsyst.lsystem` définies dans le fichier `lsystem.scm`, ainsi que de développer les représentations des L-systems de la section 4.

Remarque : Deux objets sont définis dans `lsystem.scm`. D'une part, il y le `Turtle L-system` et d'autre part le `L-system`. Ce sont des objets différents ! En particulier, le `Turtle L-system` contient l'information d'angle nécessaire aux dessins.

Conseil : il peut être opportun de ne pas calquer la représentation Scheme du L-system sur sa définition formelle.

6.2 `turtle.scm`

Il s'agit d'implémenter la fonction `cons-init-turtle-cont`, correspondant à l'interpréteur du langage tortue.

1. Voir, par exemple, `https://sarasoueidan.com/blog/svg-coordinate-systems/`.

6.3 `svg.scm`

Il s'agit d'implémenter la fonction `drawing->svg`, correspondant à l'enregistrement de la figure au format SVG.

7 Rapport

En plus de l'implémentation, il vous est demandé de soumettre un rapport au format PDF reprenant les réponses aux questions suivantes :

1. Proposez un L-system pour le flocon de Koch (`https://en.wikipedia.org/wiki/Koch_snowflake`)
2. Décrivez la représentation que vous avez utilisée pour les L-systems ainsi que vos choix d'implémentation pour la génération des chaînes.
3. Décrivez vos choix d'implémentation pour l'interpréteur des chaînes.
4. Décrivez vos choix d'implémentation pour l'enregistrement au format SVG.

N'hésitez pas à proposer d'autres L-systems.

8 Deadline

Le projet est à réaliser par groupe de **trois** étudiants et est à rendre pour le **23 avril 2021**.

Vous devez rendre une archive au format `tar.gz` comprenant votre rapport, ainsi que les trois fichiers `lsystem.scm`, `turtle.scm`, `svg.scm`.

Toutes les soumissions se feront via la plateforme `http://submit.montefiore.ulg.ac.be` et doivent tourner sur les machines ms8xx.

Afin d'anticiper tout problème liés à la formation des groupes, vous devez constituer ceux-ci sur la plateforme de soumission pour le **23 mars 2021**. Passé cette date, il ne sera plus possible de constituer un groupe, et donc de soumettre le projet.

N'oubliez pas de **spécifier toutes les fonctions auxiliaires** (même celles définies à l'aide d'un `letrec`) ainsi que toutes **les structures** que vous implémentez.

Pour rappel, le plagiat est sévèrement sanctionné.

Bon travail