L-Système

Avant de commencer

Pour ce TP nous avons besoin du module graphique `gloss`, vous pouvez l'installer grâce à la ligne de commande :

```
stack install gloss
```

Ajoutez également au début de votre fichier la ligne suivante :

```
import Graphics.Gloss
```

Un module de tortue !

Le module `turtle` n'existe pas en Haskell. Il y a bien un module qui porte ce nom, mais cela n'a rien à voir avec ce que nous voulons faire. Ce n'est pas grave, on est des pros de la programmation fonctionnelle maintenant, on va donc implémenter notre propre module `turtle` !

Une tortue est caractérisée par 2 informations:

- Sa position : représenté par le type `Point` qui est défini dans le module `Gloss`. Un point est un tuple à 2 éléments correspondant aux coordonnées .
- Son orientation : C'est un angle par rapport à l'axe des abscisses. (**Exemple**: `0` la tortue regarde vers la droite, `90` vers le haut, etc.)

1. Définissez un type `etatTortue` étant un tuple avec ses trois informations.

2. Écrivez une fonction `avance :: Float -> EtatTortue -> EtatTortue` qui calcule l'état de la tortue après avoir avancé de `a` pixels.

Note: petit rappel de trigonométrie pour calculer la nouvelle position de la tortue

- $x' = x + a * cos(orientation)$
- $y' = y + a * sin(orientation)$

Où *orientation* est un angle en radian. Pour rappel, la formule pour convertir un angle en degré en radian, il faut utiliser la formule suivante : $(angle * \pi)/180$. Vous pouvez utiliser la constante `pi` déjà définie dans Haskell.

Maintenant, nous allons tester notre fonction pour afficher le déplacement de la tortue. Mais avant ça, nous avons besoin d'écrire encore quelques fonctions :). Le module `Gloss` permet de dessiner un chemin grâce à la structure de données `Picture` (Documentation). Nous allons utiliser le constructeur `Line` qui prend un `Path` en paramètre. Un `Path` est une liste de `Point` qui représente le chemin à tracer.

Un exemple vaut mieux que 1000 mots, ajoutez la fonction main ci-dessous dans votre fichier :

```haskell
main = display (InWindow "Exemple" (600, 400) (0, 0)) white (Line [(0.0,0.0),(100.0,100.0)])
```

Cette fonction va dessiner un chemin à 2 points $(0; 0)$ et $(100; 100)$. Vous pouvez tester cette fonction en l'appelant dans l'interprète.

3. Définissez une fonction `ajouterPoint :: Path -> EtatTortue -> Path` qui ajoute un point au `Path` en fonction de l'état de la tortue.

Si tout est correct, le code si dessous doit vous faire un trait de 100 pixels.

```haskell
etat = ((0.0,0.0), 0)
chemin = ajouterPoint (ajouterPoint [] etat) (avance 100 etat)
main = display (InWindow "Tortue" (600, 400) (0, 0)) white (Line chemin)
```

4. Définissez une fonction `tournerAGauche :: Float -> EtatTortue -> EtatTortue` qui tourne la tortue vers la gauche d'un certain angle.
5. Définissez une fonction `tournerADroite :: Float -> EtatTortue -> EtatTortue` qui tourne la tortue vers la droite d'un certain angle.

Vous pouvez tester vos fonctions à l'aide du code ci-dessous par exemple :

```
etat = ((0.0,0.0), 0)
chemin = ajouterPoint (ajouterPoint [] etat) (tournerAGauche 45 (avance 100 etat))
main = display (InWindow "Tortue" (600, 400) (0, 0)) white (Line chemin)
```

On a tout ce qu'il faut pour la suite du TP, donc ce sera tout pour notre module Tortue.

Les L-Systèmes

M. Weinberg m'a dit que vous connaissiez les L-Systèmes, alors on va manipuler des L-Systèmes en programmation fonctionnelle !

Un L système est défini par :

1. Un *alphabet* fini de symboles
2. Des *règles de dérivation*
3. Un *axiome*

Exemple

Voici un L-système permettant d'obtenir le flocon de Von Koch :

1. L'Alphabet : $A = \{F, +, -\}$, où F indique d'avancer et + ou - de tourner à droite ou gauche
2. Les règles de dérivation :
    1. $F \to F - F + +F - F$
    2. $+ \to +$
    3. $- \to -$
3. L'axiome $u_0 = F$

Les règles de dérivation indiquent ce par quoi il faut remplacer chaque symbole. Exemple sur les premières itérations du Flocon de Von Koch :

Rang	u_n
0	F
1	F-F++FF
2	F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F

Mise en œuvre

Définition de type

La première étape est de définir les types dont on aura besoin pour manipuler correctement les L-Systèmes

6. Définissez un type `Symbole` qui est une simple redéfinition du type `Char`
7. Définissez un type `Mot` qui est une liste de `Symbole`
8. Définissez un type `Axiome` qui est un `Mot`
9. Définissez un type `Regle` qui est une application prenant un `Symbole` et donnant un `Mot`
10. Définissez un type `LSysteme` qui est une liste de `Mot`.

Dérivation

11. Définissez une fonction `reglesVonKoch :: Regles` qui reprends les règles du Flocon de Von Koch

12. Définissez une fonction `motSuivant :: Regles -> Mot -> Mot` qui calcule le mot suivant à partir d'un mot et d'un ensemble de règles de dérivation.

13. Définissez une fonction `lSysteme :: Axiome -> Regles -> LSysteme` qui calcule le L-Système. **Remarque**: Le L-Système est une liste infinie.

On a maintenant un module de Tortue, et de quoi faire des L-Systèmes, il n'y a plus qu'à combiner les deux !

Interprète L-Système

Dans notre L-Système, les déplacements et rotations sont toujours les mêmes. On va donc créer un type `Config` qui contiendra les informations de déplacement de notre Tortue.

```haskell
type Config = (EtatTortue, -- Etat initial de la tortue
               Float,      -- Longueur de déplacement
               Float)      -- Angle de rotation de la tortue
```

14. Définissez les fonctions ci-dessous, qui sont de simples accesseurs :

- `etatInitial :: Config -> EtatTortue`
- `longueurPas :: Config -> Float`
- `angleRotation :: Config -> Float`

Lorsque la tortue va bouger, son état va changer. À la fin de chaque déplacement, il faut également ajouter un point au chemin. Ces deux informations seront représentées par un tuple dont le nom de type sera `EtatDessin`.

15. Définissez le type `EtatDessin`.

16. Définissez la fonction `interpreteSymbole :: Config -> EtatDessin -> Symbole -> EtatDessin` qui interprète un symbole et modifie l'état du dessin.

17. Définissez la fonction `interpreteMot :: Config -> EtatDessin -> Mot -> EtatDessin` qui interprète un mot.

18. Définissez une fonction `cheminLSysteme :: Config -> LSysteme -> Int -> Path` qui retourne le chemin d'un L-Système à un rang passé en paramètre.

19. Modifiez la fonction `main` pour afficher le flocon de Von Koch au rang 4. Avec la configuration suivante :

```haskell
vonKoch = (((-15,0),0),10,60)
```

Voici le résultat que vous devez obtenir :

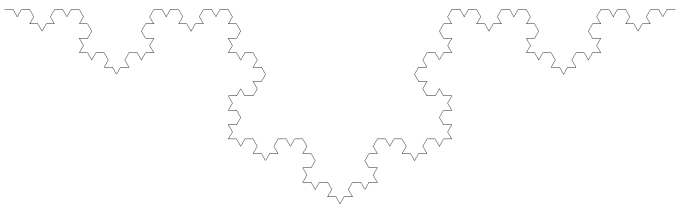

Figure 1: Flocon de VonKoch