Dessiner avec une tortue

Master 2 I2L, 2017/2018

Le but de ce travail est de mettre en place une tortue de type logo. Cet animal robotique est capable de se déplacer en recevant des ordres. Elle est aussi munie d'un crayon qui peut laisser la trace de son déplacement.

Dans premier temps, nous mettons en place une sortie graphique à Haskell pour être capable de dessiner des formes géométrique simple. Dans un deuxième temps, nous créons notre tortue logo à l'aide de fonctions de base. Enfin, nous commencerons à définir une syntaxe abstraite d'un langage capable de commander notre tortue.

1 Ecran graphique

Le langage Haskell ne possède pas de sortie graphique qui permette de dessiner simplement des formes géométriques telles que des carrés, cercles, lignes, etc. Il serait peut-être possible de définir une interface graphique avec la bibliothèque wx. Mais cette bibliothèque est "lourde" pour seulement le dessin que l'on désire.

Il est donc nécessaire de faire appel à un autre programme pour pouvoir dessiner. Plusieurs niveaux de couplage entre Haskell et ce programme sont alors envisageables :

- Couplage fort : la bibliothèque logicielle externe est compilée et directement accessible depuis le langage Haskell.
- Couplage intermédiaire : la communication s'effectue par l'intermédiaire d'une socket utilisant le réseau.
- Couplage faible : la communication entre les deux programmes par l'intermédiaire de fichiers.

Dans ce travail nous choisirons un couplage faible avec une communication à l'aide de fichiers. Cette solution, la plus simple, est satisfaisante pour l'affiche d'un dessin statique.

Le programme qui permet l'affichage sera un navigateur web. En effet, il est possible d'afficher des images au format svg à l'aide d'un navigateur. Ce format permet de dessiner simplement au format xml des figures géométriques simples.

1.1 Mise en place

a- Créer une image au format svg directement avec un éditeur de texte. Dessiner au moins un rectangle, un cercle, un segment, et une ligne brisée. Changer la couleur de ces figures.

Il n'est pas possible d'afficher un fichier svg qui n'est pas totalement correct. On ne peut donc pas créer ce fichier "à la volée" depuis Haskell sans provoquer une erreur. La solution consiste alors de représenter une image dans le langage Haskell (à l'aide de tuples et listes), puis de définir une fonction "export" qui permet de créer le fichier svg depuis la représentation Haskell. Cette représentation est alors un type de données abstrait (TDA) d'une image dont il faut définir les

primitives (type de base, écriture et lecture).

b- Définir une représentation Haskell, *i.e.* un TDA, d'une image qui peut au moins contenir un rectangle, un cercle, un segment et une ligne brisée.
c- Définir la fonction export qui crée le fichier svg à partir de la représentation Haskell.

1.2 Quelques dessins

d- Définir une fonction qui dessine sur l'écran des carrés dont la couleur et la position sont aléatoires.
e- Réaliser l'affichage de la dynamique d'un automate cellulaire du TP 05.

2 La tortue

Une tortue est définie comme dans le langage logo. Elle porte un crayon qui peut être levé ou baissé. Elle peut également avancer ou reculer, tourner à droite d'un angle donné, ou tourner à gauche. Lorsque le crayon est baissé, la tortue laisse la trace de son déplacement sur l'écran graphique.

L'écran joue le rôle d'environnement pour la tortue. En effet, lorsque la tortue se déplace, l'écran est modifié. Lorsque la tortue apparait pour la première fois, elle est situé au milieu de son environnement. L'écran joue le rôle d'environnement dans lequel la tortue évolue. Le "monde" est donc constitué de la tortue ainsi que de l'écran graphique, son environnement.

Un utilisateur peut donner des ordres à la tortue en les énonçant dans ce monde. Les ordres correspondent aux capacités de la tortue : tourner, avancer, reculer, lever ou baisser le crayon. Un ordre permet également de faire naitre une tortue au milieu de son environnement et d'effacer l'écran.

Informatiquement, le monde est codé par un couple :

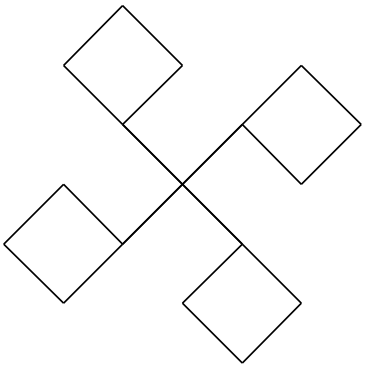
```
( tortue, ecran )
```

Question :
— Définir les fonctions permettant d'animer la tortue décrite ci-dessus.

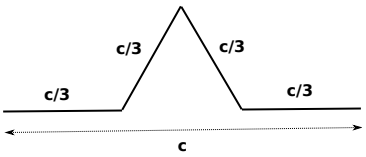
3 Dessiner avec une tortue

En donnant des ordres à notre tortue, il est maintenant de réaliser des dessins simplement.

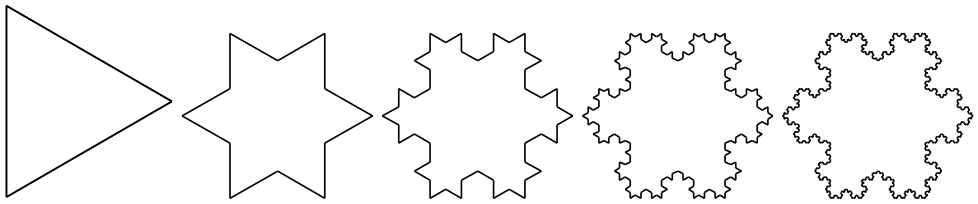
a- Définir une fonction qui permet de faire dessiner un carré à la tortue.
b- Définir une fonction qui permet de faire dessiner un polygone régulier à la tortue. Dessiner ainsi une approximation de cercle.
c- Définir une fonction qui permet de faire dessiner un moulin comme ceci :



d- Définir une fonction pour dessiner le flocon de Von Kock (voir ci-dessous). Ce flocon est un dessin récursif. A la génération 0, le motif est simplement une ligne de longueur c. A la génération n, le motif est constitué de 4 morceaux comme ceci (attention aux angles) :



Chaque morceau est lui-même un motif de la génération $n-1$.
Vous pourrez obtenir de beaux flocons (de gauche à droite, de la génération 0 à la génération 4) :



4 Langage de la tortue

Dans cette partie, nous allons définir une syntaxe abstraite qui permet de contrôler notre tortue logo. Mais d'abord ajoutons une autre fonctionnalité à notre tortue.

La tortue logo possède une autre capacité, celle de pouvoir repérer une série d'ordres. Par exemple, elle peut exécuter :

```
répéter 4 fois [
    avancer 50
    tourner 90
]
```

Questions :

a- Ajouter une fonction qui permet de répéter une série d'ordres.

b- En utilisant la fonction précédente, dessiner de nouveau un carré.

Comme vous pouvez vous en rendre compte, il vaut mieux être informaticien pour donner des ordres à la tortue. En particulier, lorsqu'il s'agit de répéter des ordres. Il serait plus facile de donner les ordres sous forme de liste de la manière suivante :

```
[ av 50, td 90, av 50, td 90, av 50, td 90, av 50, td 90 ]
```

ou encore

```
[ repeter 4 [ av 50, td 90 ] ]
```

Ou en utilisant une syntaxe d'Haskell :

```
[ Av 50, Td 90, Av 50, Td 90, Av 50, Td 90, Av 50, Td 90 ]
```

et

```
[ Repeter 4 [ Av 50, Td 90 ] ]
```

Questions :

c- Définir la fonction `executer_ordre` capable d'exécuter un seul ordre simple (tout sauf l'ordre répéter)

d- Définir la fonction `executer_liste_ordres` capable d'exécuter une liste d'ordres simples.

e- Ajouter l'ordre `répéter` à la fonction `executer_ordre`.

f- Vous pouvez maintenant définir la fonction `executer_programme` qui exécuter un programme écrit sous forme d'une liste d'ordre.

5 Pour aller plus loin

Notre petit langage n'a pas de notion de variable ni de fonction.

a- Ajouter une notion d'expression arithmétique contenant des variables.
De cette manière nous pourrions écrire dans la syntaxe abstraite :

```
[ soit a 5, repeter (a - 1) [ av 50, td 90 ] ]
```

b- Ajouter la notion de fonction au langage.