2009-2010
Université Paris Diderot
Licence 2 Mathématiques Informatique
Licence 3 Informatique
Master 1 Linguistique informatique

Projet de programmation fonctionnelle
Labyrinthes

Notes préliminaires

Le projet est à faire impérativement par groupe de 2 (avec une tolérance pour les monômes, si vous venez nous en parler avant).

Vous devrez rendre votre projet par mail sous la forme d'un fichier `.tar.gz` qui contiendra l'intégralité du code ainsi qu'un fichier `INSTALL` contenant en quelques lignes les indications pour compiler (en utilisant make ou ocamlbuild) et exécuter votre programme. Votre programme doit être fait en OCaml et doit se présenter sous la forme d'un exécutable (pas au toplevel !) avec des arguments, permettant de choisir notamment quelle version du programme doit être exécutée.

Le projet est en trois parties de taille sensiblement égales. L'énoncé est très détaillé au début, beaucoup moins à la fin, afin de vous laisser beaucoup de liberté. *La première partie est à rendre pour le 1er décembre* par email (sans rapport). La troisième partie est une extension au choix. L'intégralité du projet (qui pourra contenir des modifications de la première partie) sera à rendre par email début janvier avant la soutenance, avec un rapport. La date exacte sera précisée sur la page Web du cours.

Le rapport doit détailler précisément les types utilisés et les choix des programmation que vous avez faits avec leur justification (organisation du code, complexité des algorithmes, encapsulation et abstraction des types, etc.). Le rapport ne doit pas contenir le code du projet.

Lors de la soutenance, une démonstration des trois parties du projet vous sera demandée. Les soutenances auront lieu par groupe mais les notes seront individuelles. Vous devrez donc faire très attention à vous répartir le temps de parole.

L'utilisation de traits impératifs doit être limité au strict minimum et justifié de manière précise dans le rapport. Le code doit comporter plusieurs modules et les interfaces doivent être définies proprement. La note finale tiendra largement compte de cette séparation ainsi que de la définition et abstraction des types.

Introduction

Dans ce projet, nous allons dans un premier temps construire des labyrinthes dit *parfaits*, puis les afficherons, et enfin les résoudrons automatiquement, éventuellement de manière intelligente.

Un labyrinthe est dit *parfait* lorsqu'il y a un unique chemin reliant tout couple de points. Afin de réaliser un tel labyrinthe, nous allons utiliser l'algorithme suivant (illustré par la figure 1) :

1. Fermer toutes les cases du labyrinthe (par quatre murs dans le cas d'un labyrinthe carré, six dans un labyrinthe en nid d'abeilles), et les colorier toutes de façons différentes.
2. Choisir un mur au hasard entre deux zones (donc entre deux cases de couleurs différentes) et le supprimer. On appellera cela une *porte*. On coloriera la seconde zone de la même couleur que la première.
3. Répéter l'operation jusqu'à ce que le labyrinthe ne comporte plus qu'une seule couleur.

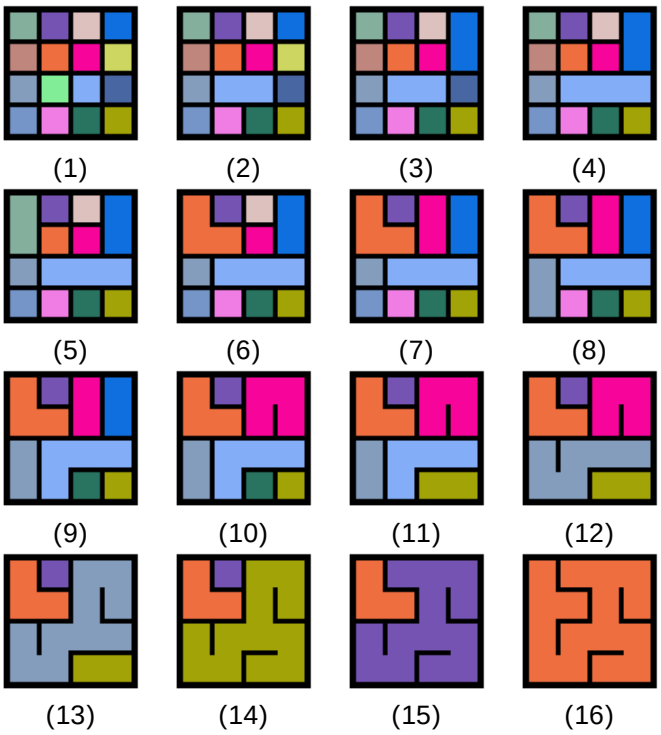

FIG. 1 – Exemple de déroulement

Avec cet algorithme, on est assuré de ne faire communiquer deux parties du labyrinthe que par une seule porte, et on a bien un labyrinthe parfait. Bien entendu, on pourra implanter les couleurs de l'algorithme par n'importe quel type dénombrable.

Partie 1

1 – Types et primitives

Question 1

Nous allons commencer par des labyrinthes à cases carrés. Un labyrinthe est alors une matrice de cases. Nous modéliserons une case par sa couleur et la liste de ses portes déjà ouvertes. Une porte peut donc prendre quatre valeurs (directions) distinctes.

Écrivez les types `door`, `cell` et `labyrinth`. Vous pouvez éventuellement étendre ces types de toutes les informations qui vous semblent utiles pour écrire vos algorithmes.

Question 2
Écrivez la fonction `new_labyrinth` *width height* qui crée un labyrinthe dans l'état initial de l'algorithme.

Question 3
Donnez les fonctions utilitaires sur les portes :
- `open_door` *laby x y door*
  : *labyrinth* → *int* → *int* → *door* → *unit*
- `door_opened` *laby x y door*
  : *labyrinth* → *int* → *int* → *door* → *bool*
- `door_closed` *laby x y door*
  : *labyrinth* → *int* → *int* → *door* → *bool*

Question 4
Donnez les fonctions
- `neighbour` *x y door*
  : *int* → *int* → *door* → *(int * int)*
  donnant les coordonnées de la case voisine en passant par la porte donnée,
- `opposite` *door*
  : *door* → *door*
  donnant la porte de la case voisine par laquelle on arrive.

Utiliser ces fonctions permet de s'abstraire du type `door` et il suffira de les changer pour modifier la forme et la disposition des cases du labyrinthe.

Donnez aussi la valeur `all_doors` : *door list* donnant toutes les portes possibles pour une case.

Question 5
Pour compléter l'abstraction, définissez la fonctionelle `choose_door` *laby op*
: *labyrinth* → *(labyrinth* → *int* → *int* → *door* → α*)* → α
qui choisit une porte valide (débouchant à l'intérieur) du labyrinthe et la transmet à l'opération *op*.

2 – Génération
Maintenant que nous avons des primitives abstraites sur les cases, nous allons implanter l'algorithme de génération. Les fonctions de cet exercice ne devront donc pas faire apparaitre le type `door` directement.

Question 1
Donnez la fonction `change_color` *laby x y c* qui devra remplir la zone contenant la case de coordonnées x, y

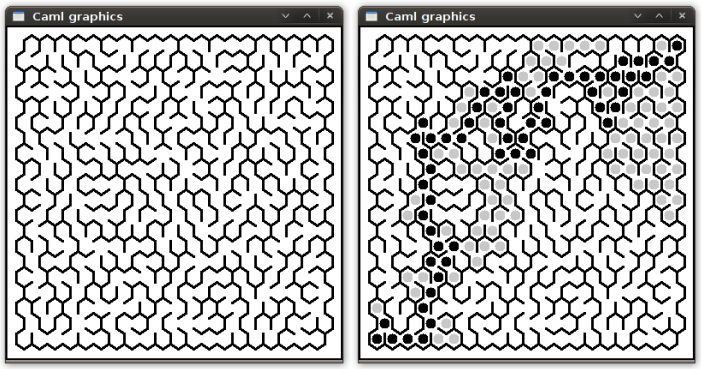

FIG. 3 – Labyrinthe en nid d'abeilles

Question 1
Identifiez les types et fonctions nécessitant une modification.

Question 2
Proposez une achitecture permettant de construire un programme unique qui sera capable de traiter le cas des labyrinthes à cases carrées et en nid d'abeilles. Vous veillerez à dupliquer le moins de code possible et à abstraire les types quand ce sera utile. Modularisez votre programme et construisez un exécutable.

Partie 3 - Extension

Vous devez réaliser une extension au choix pour le projet, selon votre imagination. Cette extension pourra nécessiter la modification des types de données et algorithmes du projet, mais vous devrez d'abord présenter le projet sans l'extension et respectant exactement les directives de l'énoncé.