Cycle ingénieur - 2ème année GSI

Programmation fonctionnelle en Scala

Projet *Property testing*

2022-2023

Consignes

Consignes générales

Objectif du projet

- Mettre en place une API simplifiée de *property testing*

Conditions du projet

- Projet effectué par groupes de **4 à 5 étudiants**
- Date limite de rendu (par mail) : **2 avril 2023, 23h59**
- Soutenance : semaine du **3 au 7 avril 2023**

Consignes générales : nature du rendu

Fourni

- Quatre modules : `Property`, `Generator`, `Reduction` et `Test`
- Quelques exemples d'utilisation

À faire

1. Implémenter les éléments de chaque module
2. Proposer une ou plusieurs fonctionnalité(s) supplémentaire(s)
3. Proposer d'autres exemples d'utilisation

À rendre : archive contenant

- les quatre modules
- les exemples
- rapport court

Consignes spécifiques

Modification du code existant

- Modification **INTERDITE** :
  - de la signature des valeurs et fonctions à implémenter
  - des (quelques) éléments déjà implémentés
- Ajout *autorisé* de nouveaux éléments intermédiaires pour implémenter les éléments demandés
- Ajout *autorisé* de nouvelles fonctionnalités

Tests des valeurs et méthodes à implémenter

- Des tests seront effectués sur votre code.
  Ces tests participent à la note finale du projet.
- Pour fonctionner, **ces tests supposent que les consignes précédentes ont été respectées**. *Sinon, ces tests ne fonctionneront pas.*

Consignes spécifiques

Respect des règles de programmation fonctionnelle

- Utilisation de variables **interdite**
- Utilisation des boucles (conditionnelles ou non) **interdite**

Quelques conseils

- **Réduire le recours à la récursivité** en utilisant les fonctions de la librairie standard OCaML
- sinon **favoriser la récursivité terminale**

Rapport

Description

- Description succincte de la solution mise en place
  (y compris la ou les fonctionnalité(s) supplémentaire(s))
- Difficultés rencontrées **et leur résolution**

Analyse

- Pertinence des ou de la fonctionnalité(s) proposée(s)
- Avantages
- Limites

Soutenance (semaine du 3/4 au 7/4)

Organisation

- Durée totale : *20 minutes*
  - Temps de présentation : **entre 10 et 15 min**

    Contrainte de temps à respecter absolument
  - Reste du temps consacré à nos questions
- **Tous les membres du groupe doivent présenter.**

Attendu

- Présenter la solution implémentée
  (y compris la ou les fonctionnalité(s) supplémentaire(s))
- **Présenter des exemples pour valider l'implémentation**

Présentation du projet

Présentation générale

Motivation : *QuickCheck* (bibliothèque Haskell)

1. Définir une propriété devant être vérifiée par une (ou plusieurs) fonction(s) quels que soient les paramètres
2. Générer *aléatoirement* des paramètres de test
3. Si un jeu de paramètres ne vérifie pas la propriété, en chercher un plus « simple » ne la vérifiant pas.

Composition de l'API (générique)

- `Property` : gestion des propriétés
- `Generator` : génération pseudo-aléatoire de données
- `Reduction` : stratégies de simplification des contre-exemples
- `Test` : gestion des tests

Module Property

```
type 'a Property.t = 'a -> bool
```

Fonctionnalités à implémenter

- Cas particuliers : toujours vrai, toujours faux

Indications

- Paramètre du type générique = type des éléments sur lesquels portent la propriété

Module Generator

Fonctionnalités à implémenter

- Définition du type générique
- Application du générateur pour générer une valeur
- Générateurs de types de base (booléen, entier, flottant, etc...)
- Générateurs de chaînes, de listes
- Transformations, filtrage, etc...

Indications

- Paramètre du type générique = type des valeurs à générer
- Module Random déjà existant :
  - s'en inspirer pour définir le type générique
  - utiliser les générateurs de ce module pour construire les générateurs de types de base demandés

Module Reduction

```
type 'a Reduction.t = 'a -> 'a list
```

Fonctionnalités à implémenter

- Stratégie « vide » (ne fournissant aucune suggestion)
- Stratégie pour les types de base (`int`, `float`, etc...)
- Stratégie pour les chaînes, les listes
- Transformation : couplage

Module Reduction

Indications

- Paramètre du type générique = type sur lequel porte la stratégie
- **Ne pas implémenter la stratégie vide impliquera une non-validation de tous les tests du module suivant (`Test`).**
- Ne **jamais** proposer la valeur donnée en paramètre !
- Essayer de lier une stratégie au générateur correspondant
  (cf. documentation dans le code fourni)
- Une stratégie ne dépend pas d'une propriété
  ⟹ elle fournit uniquement des *suggestions*, **en commençant par la plus « simple »**
  ⟹ fournir *plusieurs* suggestions donne plus de chance que certaines puissent être applicables

Module Test

Fonctionnalités à implémenter

- Définition du type générique
- Création d'un test
- Lancement d'un test :
  - Vérifier si le test réussit ou non
  - Recherche d'un contre-exemple « simple » (si possible)
- Lancement de plusieurs tests

Indications

- Paramètre du type générique = type des **paramètres** de la propriété à tester
- Un test consiste à vérifier une propriété en s'appuyant sur :
  - un générateur pseudo-aléatoire pour générer les données de test ;
  - une stratégie de réduction pour rechercher des contre-exemples plus « simples ».

Exemples fournis

Division euclidienne entière

- `a = (a / b) * b + (a mod b)` ✓
- `a = (a / b) * b - (a mod b)` ✗

⟹ *portent sur des couples d'entiers*

Concaténation de listes

- `List.length (l1 @ l2) = (List.length l1) + (List.length l2)` ✓
- `List.length (l1 @ l2) = (List.length l1) * (List.length l2)` ✗

⟹ *portent sur des couples de listes*
(testées sur des listes d'entiers)