Projet OCaml
Rapport

Par Paul Lafourcade, Lorrain Barrabé, Enzo François
et Louis Floreani (ING2 GSI Groupe 1)

2 avril 2023

Projet OCaml
Rapport

Table des matières

Projet OCaml
Rapport

1 Introduction

En GSI, le module Programmation Fonctionnelle nous apprend les concepts clés de la programmation fonctionnelle à travers le langage OCaml. Ce rapport porte sur notre projet OCaml réalisé dans le cadre de ce module.

Le projet consiste à mettre en place une API simplifiée de property testing qui sera constituée de quatre modules : Property, Generator, Reduction et Test. Le but étant de permettre une vérification automatique de la qualité des programmes informatiques en générant des données aléatoires de test et en vérifiant si les propriétés attendues sont respectées.
Pour cela, des exemples d'utilisation sont fournis et plusieurs tâches sont à réaliser. Tout d'abord, il faut implémenter les éléments de chaque module. Ensuite, il est demandé de proposer une ou plusieurs fonctionnalités supplémentaires pour améliorer l'efficacité et la flexibilité de l'API. Enfin, il fallait proposer d'autres exemples d'utilisation pour mettre en avant les capacités de cette API.
De plus, le projet a respecté les règles de programmation fonctionnelle en évitant l'utilisation de variables et de boucles. Nous avons préféré utiliser les fonctions de la librairie standard d'OCaml plutôt que d'utiliser la récursivité. Cependant, nous pouvions utiliser la récursivité lorsque celle-ci était nécessaire, mais il nous était recommandé d'utiliser la récursivité terminale.

Il est important de noter qu'il était interdit de modifier la signature des valeurs et fonctions à implémenter ainsi que certains éléments déjà implémentés. Toutefois, l'ajout de nouveaux éléments intermédiaires pour implémenter les éléments demandés et de nouvelles fonctionnalités est autorisé.

2 Description

2.1 Description succinte de la solution mise en place (fonctionnalité(s) supplémentaire(s) comprise(s))

Cette API se compose de quatre modules : Property, Generator, Reduction et Test.

- Pour le module Property, il fallait implémenter les méthodes always_true et always_false, mais nous avons ajouté quatre fonctionnalités au module Property, qui sont les fonctions not_prop, and_prop, or_prop, et implies_prop.
Ces quatre fonctions permettent de manipuler des propriétés. La fonction not_prop retourne la négation d'une propriété donnée, and_prop retourne la conjonction de deux propriétés données, or_prop retourne la disjonction de deux propriétés données et im-

plies_prop retourne l'implication de deux propriétés données. Grâce aux fonctionnalités ajoutées dans le module Property, le nombre de possibilités d'utilisation du module Property se voit augmenter.

- Pour le module Generator, il fallait définir le type générique, créer des générateurs pour les types de base et les collections, et utiliser le module Random pour construire les générateurs de types de base ou s'en inspirer pour définir le type générique.

- Pour le module Reduction, il fallait implémenter des stratégies pour les types de base, les collections et ne jamais proposer la valeur donnée en paramètre. Il nous fallait aussi implémenter la stratégie vide. Ces stratégies ne dépendent pas d'une propriété mais fournissent des suggestions.

- Pour le module Test, il fallait créer un test et le lancement d'un test. Ces tests consistent à vérifier une propriété en s'appuyant sur un générateur pseudo-aléatoire et une stratégie de réduction pour trouver des contre-exemples plus simples.

- Enfin, dans les exemples, nous avons ajouté l'addition de deux entiers, la multiplication de deux flottants, la concaténation de deux chaînes de caractères et la concaténation de listes de caractères, afin d'élargir le nombre de fonctionnalités de l'API testée.

2.2 Difficultés rencontrées et leur résolution

Le module qui a présenté le plus de difficultés lors de son implémentation est le module Reduction. Nous avons d'abord eu beaucoup de mal à bien saisir le sens de ce module. Une fois tous les autres modules implémentés, nous avons dû nous focaliser sur celui-ci, après avoir déjà bien réfléchi en amont sur le sens des fonctions qui le composaient. Nous avons mis beaucoup de temps à coder ce module, puis à faire en sorte qu'il compile, mais cela a eu un grand impact sur la durée de la phase de test de celui-ci dans le fichier exemples.ml.

Nous avons également eu des problèmes liés à la compilation du code que nous n'avions pas envisagés. De fait, nous nous sommes concentrés sur leur résolution, afin de pouvoir proposer un programme qui compilait, et dont tous les modules étaient soigneusement implémentés. Malheureusement, cela s'est fait au détriment d'autres parties notamment celle de l'étoffement des fonctionnalités de l'API.

De plus, lors de la compilation des exemples, sur les tests de concaténation de listes de caractères et de concaténation de deux chaînes de caractères, nous avons eu une erreur de Stack Overflow, sûrement dû à un problème de récursivité. Pourtant, ces codes fonctionnaient, mais vers les dernières vérifications de notre code cette erreur est apparue et nous avons pas réussi à retracer son origine à temps. Ces deux tests ont donc été mis en commentaires.

Enfin, en nous en tenant à la signature des modules, nous avons eu plusieurs problèmes liés aux types de retour des fonctions. En effet, de nombreuses fois il nous est

arrivé que le type de retour de la fonction ne correspondait pas au type spécifié dans sa signature, ce qui posait des soucis lors de la compilation. Au final, après plusieurs manipulations du code et mises en commun de nos idées sur la question, nous avons réussi à régler ces problèmes mineurs.

3 Analyse

3.1 Pertinence des ou de la fonctionnalité(s) proposée(s)

Grâce aux fonctionnalités ajoutées dans le module Property, le nombre de possibilités d'utilisation du module Property (qui ne contenait alors que deux fonctions à proprement parler) se voit augmenter.

La fonction not_prop prend en entrée une propriété booléenne et retourne la négation de cette propriété. Par exemple, si la propriété entrée est "x > 5", la fonction not_prop retournera "x <= 5".

La fonction and_prop prend en entrée deux propriétés booléennes et retourne la conjonction de ces deux propriétés. Par exemple, si la propriété entrée est "x > 5" et "x < 10", la fonction and_prop retournera "5 < x < 10".

La fonction or_prop prend en entrée deux propriétés booléennes et retourne la disjonction de ces deux propriétés. Par exemple, si la propriété entrée est "x < 5" ou "x > 10", la fonction or_prop retournera "x < 5 || x > 10".

La fonction implies_prop prend en entrée deux propriétés booléennes et retourne l'implication de ces deux propriétés. Par exemple, si la propriété entrée est "x > 5" implique "x < 10", la fonction implies_prop retournera "x < 10 || x <= 5".

3.2 Avantages

Les avantages de ces fonctions sont qu'elles augmentent la flexibilité de l'API en permettant aux utilisateurs de manipuler plus facilement les propriétés et de les combiner pour effectuer des tests plus avancés. Par exemple, il est possible de combiner des propriétés pour vérifier des comportements complexes.

3.3 Limites

Les limites de ces fonctions sont que leur utilisation peut devenir compliquée pour des tests très complexes, et qu'il peut être difficile de trouver une combinaison optimale de propriétés pour atteindre un haut niveau de couverture de tests.

4 Conclusion

En conclusion, ce projet nous a permis de mieux comprendre les concepts clés de la programmation fonctionnelle à travers le langage OCaml, et de mettre en place une API de property testing pour vérifier automatiquement la qualité des programmes informatiques.
Nous avons rencontré des difficultés lors de l'implémentation du module Reduction, mais nous avons résolu ces problèmes pour proposer un programme qui compilait et dont tous les modules étaient soigneusement implémentés. Finalement avons respecté les règles de programmation fonctionnelle, ajouté des fonctionnalités supplémentaires et proposé des exemples d'utilisation pour mettre en avant les capacités de cette API.