Soyez prudent : prenez des photos pour l'assurance avec osnap (démonstration)

Valentin Chaboche, Zaynah Dargaye, Arvid Jakobsson

Soyez prudent : prenez des photos pour l'assurance avec `osnap`

Valentin Chaboche, Zaynah Dargaye et Arvid Jakobsson

Nomadic Labs, Paris, France `prénom.nom@nomadic-labs.com`

Résumé

Comment s'assurer que nous n'introduisons pas de bogue durant l'évolution d'un logiciel ? Les outils de "snapshot testing" offrent une solution : le résultat d'une fonction est capturé et après modification comme de la refactorisation de code, l'outil détectera des changements. Néanmoins, le snapshot testing oblige le développeur à écrire des scénarios à la main. Nous introduisons `osnap` : une bibliothèque de snapshot testing avec une génération aléatoire de scénarios, d'exécution et de détection de changement de comportement – inspirée par la génération aléatoire de bibliothèque comme QuickCheck en Haskell. En utilisant `osnap`, les développeurs peuvent sans effort générer massivement des tests de régression.

1 Introduction

Les tests de régressions sont un ensemble de tests d'un logiciel préalablement développé, qui après modification, s'assure que les modifications n'introduisent pas des défauts dans le programme. Le snapshot testing est une technique particulière de test de régression : les snapshots sont des artéfacts représentant la capture d'événements ou comportements de programmes. Sa particularité réside dans l'exécution d'une comparaison entre les snapshots avant et après modifications.

Il existe en OCaml la bibliothèque `ppx_expect` [6] qui nous permet d'ajouter directement dans le code le résultat textuel attendu d'un test. Lors de son exécution, une différence textuelle est appliquée sur l'actuel résultat et celui attendu. En revanche, chaque test doit être généré en fournissant des instances unitaires des entrées du programme. Les tests pourront difficilement couvrir tous les chemins d'exécution d'un programme, et souffriront d'un biais de la part du testeur dans le choix des entrées.

`QCheck` [4] est une bibliothèque OCaml de test à base de propriété qui propose une approche différente de test. Elle permet de tester des propriétés sur des programmes face à de nombreuses entrées générées aléatoirement. Elle permet de valider des propriétés sur des programmes en évitant donc le biais dans la sélection des entrées à contrario du test unitaire.

Nous souhaitons alors nous inspirer de la gestion de snapshot de `ppx_expect`, mais en réduisant le biais et en ayant une meilleure couverture de code grâce aux générateurs de `QCheck`. Nous pourrons alors générer un grand nombre de snapshots pour un programme de référence, qui deviendront alors nos tests de régression pour les modifications de ce programme. Ce mécanisme de test nous rappelle la bibliothèque Monolith [5] générant aléatoirement des scénarios afin de confronter une bibliothèque de référence et candidate. Cependant, nous souhaitons ici pouvoir appliquer les tests de régression à un niveau plus fin qu'une bibliothèque : des fonctions OCaml.

2 Présentation des fonctionnalités d'osnap

```
let rec expo x =
  function | 0 -> 1 | 1 -> x | n -> expo x (n-1) * x
in
let spec = Osnap.Spec.(int ^> int ^>> Result.int) in
Test.make ~spec ~count:5 ~name:"expo" expo
```

(a) L'exponentiation, sa spécification et son test osnap

```
{ "name": "expo", "scenarios": [
    expo 2 3 = 8
    expo 5 3 = 125
    expo 5 1 = 5
    expo 1 0 = 1 ] }
```

(b) Snapshot de l'exponentiation [1]

`osnap` [3] prend en entrée une spécification de signature d'une fonction, celle de la fonction à tester, cette spécification servira à la génération aléatoire de snapshot. Un snapshot est la combinaison de scénarios aléatoires appliqués à une fonction et de leurs résultats. Ces snapshots sont enregistrés sur disque afin de devenir des tests de régression.

Les spécifications d'`osnap` sont décrites grâce à un ensemble de combinateurs. Une signature pour une fonction $a_0 \to a_1 \to ... \to a_n \to b$ est composée des générateurs pour les types allant de a_0 à a_n via la bibliothèque `QCheck`. Finalement, nous voulons être capable d'observer les différences dans le résultat des différents scénarios. Il faudra alors fournir un afficheur pour le type b, afin d'appliquer une différence textuelle à la manière de `ppx_expect`.

La spécification de ces fonctions nous permet alors de générer des valeurs pour chaque paramètre d'une fonction et donc de les appliquer sur cette même fonction. Nous pouvons alors, via un test interactif fourni par `osnap`, générer `k` scénarios, que nous allons enregistrer sur disque afin de créer un snapshot, grâce à la bibliothèque `Marshal` [2].

Désormais, les tests et snapshots peuvent être versionnés et intégrés dans la suite de test. Pour chaque exécution de cette suite de test, les scénarios vont être récupérés et réappliqués sur les nouvelles versions des fonctions et vont chercher textuellement des régressions dans les retours de fonctions. Les différences observées devront alors demander une analyse du développeur pour déterminer si ces changements sont souhaités ou signifient l'introduction d'une régression.

`osnap` nous permet alors de rapidement générer un grand nombre de cas de test aléatoire via la génération de `QCheck`, contrairement aux cas unitaires de `ppx_expect`. En revanche, nous ne pouvons pas assurer des propriétés lors des changements, mais seulement détecter les éventuelles régressions dans les résultats. Cependant, nous pouvons limiter les entrées respectant des propriétés via l'écriture des générateurs. D'autre part, l'aspect statique de ces tests rend difficile les adaptations lors d'évolutions des signatures de fonctions, étroitement liées aux spécifications nécessaires à `osnap`.

Enfin, bien que la génération aléatoire des scénarios réduise le biais dans les choix d'entrée des tests unitaires, ce biais peut être introduit dans notre cas par les générateurs eux-mêmes. Nous souhaiterions alors incrémentalement améliorer la couverture du code des scénarios, afin de ne pas enregistrer en mémoire des snapshots suivant les mêmes chemins d'exécution. Ceci pourrait être fait en utilisant des outils de couverture comme `bisect_ppx` [1], et ainsi être capable de détecter des générateurs biaisés.

1. La sortie a été rendu lisible pour la compréhension

3 Conclusion

L'objectif d'`osnap` est de gagner en confiance lors des modifications d'une base de code comme des refactorisations. Les tests de régression comme technique de test permettent de détecter des changements non désirés lors de modifications. Cependant, ils nécessitent une écriture manuelle des scénarios par un développeur, et sont sujets à des biais dans l'écriture de ces derniers. Nous généralisons avec `osnap` le processus d'écriture de test de régression et snapshot testing avec la génération aléatoire de valeurs inspirée des bibliothèques de test à base de propriété comme `QCheck`.

Références

[1] Code coverage for OCaml and ReScript. URL : https://github.com/aantron/bisect_ppx.

[2] Marshaling of data structures. URL : https://ocaml.org/api/Marshal.html.

[3] Valentin Chaboche. Random snapshot testing library for OCaml. URL : https://github.com/vch9/osnap.

[4] Simon Cruanes et Rudi Grinberg et Jacques-Pascal Deplaix et Jan Midtgaard. QuickCheck inspired property-based testing for OCaml. URL : https://github.com/c-cube/qcheck.

[5] François Pottier. Strong Automated Testing of OCaml Libraries, 2021. URL : https://hal.inria.fr/hal-03049511/document.

[6] Jane Street. Expect-test - a cram like framework for OCaml. URL : https://github.com/janestreet/ppx_expect.