Test aléatoire et énumératif pour OCaml et Why3*

Alain Giorgetti[1] Jérome Ricciardi[2]
Clotilde Erard
[1] Institut FEMTO-ST, Univ. of Bourgogne Franche-Comté, CNRS, Besançon, France
alain.giorgetti@femto-st.fr
[2] Université Paris-Saclay, LMF/QuaCS, CEA/LIST/LSL, Palaiseau, France
jerome.ricciardi@cea.fr

Résumé

Nous présentons AutoCheck, un prototype d'outil de test aléatoire et énumératif de propriétés définies dans le langage fonctionnel OCaml ou dans le langage WhyML de la plateforme de vérification déductive Why3. Une originalité est que les tests énumératifs utilisent des générateurs de données eux-mêmes écrits dans le langage WhyML, et dont la correction et la complétude sont formellement prouvées avec Why3. Une autre spécificité est que l'effort de développement est réduit en exploitant le mécanisme d'extraction de Why3 vers OCaml et un outil de test aléatoire existant pour OCaml.

1 Introduction

Nous présentons la version 0.1.2 du prototype AutoCheck [EGLR21], pour tester des propriétés exécutables définies dans le langage fonctionnel OCaml [1] ou dans le langage WhyML de spécification et de programmation de la plateforme de vérification déductive Why3 [BFM+18]. AutoCheck vise l'intégration des techniques complémentaires de génération aléatoire et énumérative de données de test. Dans ce but, il exploite le mécanisme d'extraction vers OCaml de Why3, la bibliothèque ENUM de programmes d'énumération spécifiés et implémentés en WhyML et vérifiés avec Why3 [EG19], et l'outil QCheck de test aléatoire pour OCaml [CGDM20]. AutoCheck complète ce dernier avec des tests aléatoires pour les propriétés WhyML et des tests énumératifs pour les propriétés WhyML et OCaml. En s'interfaçant avec les programmes d'énumération certifiés d'ENUM, AutoCheck propose des tests énumératifs certifiés.

Ce résumé est organisé de la manière suivante. La partie 2 présente quelques aspects du test de propriétés. La partie 3 présente les principes de conception et de fonctionnement d'AutoCheck.

2 Méthodes et outils de test de propriétés

Le test de propriétés (PBT, pour *Property-Based Testing*) de programmes consiste à identifier et à tester un ensemble de propriétés que certaines fonctions doivent satisfaire. Au-delà du cas de base des contrats de fonction, qui sont des propriétés qui portent sur un seul appel d'une unique fonction, les

*Cet article est un résumé étendu d'un article [EGR22] publié le 22/02/2022 dans le *Software Quality Journal*.

1. https://ocaml.org.

propriétés relationnelles sont des propriétés qui peuvent concerner plusieurs fonctions et/ou plusieurs appels de la même fonction [BKLG+18].

Les deux approches les plus populaires pour automatiser le test de propriétés sont le test aléatoire et le test énumératif. Le test aléatoire (RT, pour *Random Testing*) utilise des générateurs pseudo-aléatoires pour produire automatiquement des cas de test. L'ancêtre des outils de test aléatoire de propriétés est QuickCheck [CH00]. Initialement écrit pour le langage Haskell, il a été adapté à plus de trente langages de programmation [2] Parmi ces outils, QCheck [CGDM20] est un outil de test aléatoire de propriétés de fonctions OCaml [MM17]. Il fournit de nombreux combinateurs utiles pour générer différents types de données, et permet également aux utilisateurs d'écrire leurs propres générateurs, en particulier pour les types inductifs. Comme la plupart des variantes de QuickCheck, QCheck fournit également une fonction de réduction (*shrinking* en anglais), qui réduit la taille du contre-exemple fourni en cas d'échec du test. Par exemple, si la propriété testée est l'absence d'un nombre donné dans une liste, elle doit fournir comme contre-exemple une liste de longueur 1 contenant uniquement ce nombre. Nous avons choisi d'intégrer QCheck dans AutoCheck, pour servir de base à un outil similaire pour WhyML. Le test énumératif (ET, pour *Enumerative Testing*) a pour objectif de générer toutes les entrées possibles des fonctions testées. Lorsque le domaine de ces entrées est trop vaste pour être exploré exhaustivement, le test exhaustif borné (BET, pour *Bounded Enumerative Testing*) n'énumère que les données de taille inférieure à une limite prédéfinie. Le test énumératif a d'abord été utilisé pour vérifier des propriétés de langages fonctionnels, un pionnier étant l'outil SmallCheck pour Haskell [RNL08]. Ensuite, il a été adapté à plusieurs assistants de preuve, par exemple à Isabelle [Bul12] et à Coq [DGG16]. Dans un travail précédent, deux des présents auteurs ont initié un outil de BET pour le langage WhyML de Why3, mais ce prototype était limité aux tableaux d'entiers [EG19].

Le test aléatoire et le test énumératif sont complémentaires. En effet, le test énumératif est utile pour les données de petite taille [DJW12], souvent peu nombreuses, mais il ne convient plus au-delà d'une certaine taille, car le nombre de données croît généralement exponentiellement avec leur taille. Le test aléatoire peut générer des données de grande taille, mais il est peu efficace pour invalider une propriété rarement fausse, tandis qu'un test énumératif borné peut soit en trouver un plus petit contre-exemple, soit prouver qu'il n'en existe aucun en dessous d'une certaine taille.

3 Architecture de l'outil

Cette partie présente l'architecture et les principes de fonctionnement du prototype AutoCheck.

La structure interne d'AutoCheck est représentée dans la figure 1. L'outil lui-même est représenté par le plus grand rectangle aux coins arrondis. Les fichiers générés automatiquement sont distingués par des rectangles en pointillés. Le code externe est indiqué en italique.

Chaque entrée d'AutoCheck est représentée par un rectangle avec des coins carrés. Il s'agit d'un fichier WhyML ou OCaml (respectivement nommé Tests.mlw ou Tests.ml dans la figure) contenant l'implémentation testée et une description des tests. Les propriétés à tester sont des fonctions OCaml ou des fonctions WhyML exécutables par extraction en OCaml, et leurs tests sont des appels de fonctions. Par conséquent, les implémentations à tester, leurs propriétés et les tests de ces propriétés peuvent être répartis dans plusieurs fichiers, comme dans n'importe quelle application OCaml ou WhyML.

2. Voir, par exemple, fast-check (`https://github.com/dubzzz/fast-check`) pour JavaScript, jetCheck (`https://github.com/JetBrains/jetCheck`) pour Java, PropEr (`https://github.com/proper-testing/proper`) pour Erlang, QuickChick (`https://github.com/QuickChick/QuickChick`) pour Coq ou theft (`https://github.com/silentbicycle/theft`) pour C.

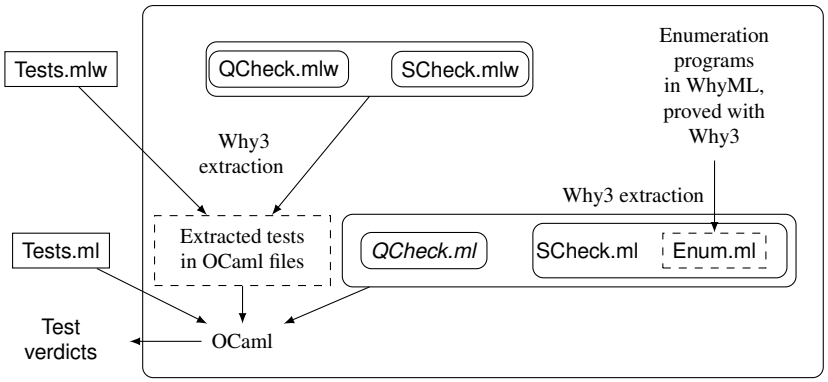

FIGURE 1 – Architecture d'AutoCheck.

Chaque test OCaml exploite un ou plusieurs générateurs de données aléatoires ou énumératifs. Les générateurs de données aléatoires sont définis dans l'outil externe de test aléatoire QCheck, dont le fichier principal est *QCheck.ml* (ce nom de fichier est écrit en italique car ce code externe est utilisé sans modifications). Les générateurs de données énumératifs sont définis dans notre outil de test énumératif pour OCaml, dont le fichier principal est SCheck.ml. Ce dernier encapsule plusieurs programmes d'énumération de la bibliothèque ENUM [EG19]. Ce code OCaml, dans le fichier Enum.ml, est automatiquement extrait par Why3 de programmes d'énumération écrits en WhyML et dont les propriétés de correction et de complétude sont prouvées avec Why3.

Les fichiers QCheck.mlw et SCheck.mlw spécifient en WhyML des fonctionnalités analogues de test aléatoire et énumératif, permettant d'écrire des tests en WhyML (dans Tests.mlw sur la figure). Leur extraction avec Why3 génère des tests en OCaml qui exploitent les outils de test aléatoires et énumératifs pour OCaml définis dans *QCheck.ml* et SCheck.ml.

4 Conclusion

AutoCheck a été conçu en privilégiant la simplicité et la facilité d'utilisation, pour ses utilisateurs et ses développeurs. D'abord, l'installation et l'utilisation sont facilitées et sécurisées par virtualisation avec Docker. Ensuite, de nombreux exemples de tests OCaml (resp. WhyML) sont fournis, dans un fichier nommé TestExamples.ml (resp. TestExamples.mlw). Ils sont ordonnés par complexité croissante et couvrent toutes les fonctionnalités de l'outil. Certains de ces exemples sont documentés dans les parties 5 et 6 de [EGR22]. Au-delà de ces exemples élémentaires, à vocation pédagogique, des exemples de tests énumératifs de fonctions et de propriétés liées aux permutations ont été présentés dans des travaux antérieurs [EG19, Gio21, EGR22].

En tant qu'outil de PBT, AutoCheck est complémentaire de l'outil Monolith [Pot21], qui implémente le test basé sur un modèle, qui doit être une implémentation de référence fournie. Côté OCaml, AutoCheck est proche de l'outil de vérification d'assertions à l'exécution **`ortac`** [FP21].

En OCaml et WhyML, les syntaxes des tests aléatoires et énumératifs ont été choisies pour être aussi similaires que possible. Une perspective est d'unifier les interfaces pour permettre d'écrire des combinaisons arbitraires de ces deux sortes de tests.

Remerciements. Ce travail est soutenu par l'EIPHI Graduate School (contrat ANR-17-EURE-0002).

Références

[BFM+18] F. Bobot, J.-C. Filliâtre, C. Marché, G. Melquiond, and A. Paskevich. *The Why3 Platform*, 2018. http://why3.lri.fr/manual.pdf.

[BKLG+18] L. Blatter, N. Kosmatov, P. Le Gall, V. Prevosto, and G. Petiot. Static and dynamic verification of relational properties on self-composed C code. In *Tests and Proofs. TAP 2018*, pages 44–62, Cham, 2018. Springer International Publishing.

[Bul12] L. Bulwahn. The new Quickcheck for Isabelle - random, exhaustive and symbolic testing under one roof. In *CPP 2012*, volume 7679 of *LNCS*, pages 92–108. Springer, Heidelberg, 2012.

[CGDM20] S. Cruanes, R. Grinberg, J.-P. Deplaix, and J. Midtgaard. QuickCheck inspired property-based testing for OCaml., 2020. https://github.com/c-cube/qcheck.

[CH00] K. Claessen and J. Hughes. QuickCheck : a lightweight tool for random testing of Haskell programs. In *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming*, volume 35 of *SIGPLAN Not.*, pages 268–279. ACM, New York, 2000.

[DGG16] C. Dubois, A. Giorgetti, and R. Genestier. Tests and proofs for enumerative combinatorics. In *Tests and Proofs. TAP 2016*, volume 6792 of *LNCS*, pages 57–75, Cham, 2016. Springer.

[DJW12] J. Duregård, P. Jansson, and M. Wang. Feat : Functional enumeration of algebraic types. *ACM SIGPLAN Notices*, 12 2012.

[EG19] C. Erard and A. Giorgetti. Bounded exhaustive testing with certified and optimized data enumeration programs. In *Testing Software and Systems. ICTSS 2019*, volume 11812 of *LNCS*, pages 159–175, Cham, 2019. Springer.

[EGLR21] C. Erard, A. Giorgetti, R. Lazarini, and J. Ricciardi. Autocheck 0.1.2, 2021. https://github.com/alaingiorgetti/autocheck.

[EGR22] C. Erard, A. Giorgetti, and J. Ricciardi. Towards random and enumerative testing for OCaml and WhyML properties. *Software Quality Journal*, 30 :253–279, 2022. https://doi.org/10.1007/s11219-021-09572-z.

[FP21] J.-C. Filliâtre and C. Pascutto. Ortac : Runtime Assertion Checking for OCaml (tool paper). In *RV'21 - 21st International Conference on Runtime Verification*, Los Angeles, CA, United States, October 2021.

[Gio21] A. Giorgetti. Théories de permutations avec Why3. In *32ème Journées Francophones des Langages Applicatifs (JFLA)*, pages 202–209, France, 2021. https://hal.archives-ouvertes.fr/hal-03190426.

[MM17] J. Midtgaard and A. Møller. Quickchecking static analysis properties. *Software Testing, Verification and Reliability*, 27(6), 2017. https://doi.org/10.1002/stvr.1640.

[Pot21] F. Pottier. Strong automated testing of OCaml libraries. In *32ème Journées Francophones des Langages Applicatifs (JFLA)*, pages 3–20, France, 2021. https://hal.archives-ouvertes.fr/hal-03190426.

[RNL08] C. Runciman, M. Naylor, and F. Lindblad. SmallCheck and Lazy SmallCheck - automatic exhaustive testing for small values. In *Proceedings of the 1st ACM SIGPLAN Symposium on Haskell*, pages 37–48. ACM, 2008.