神奈川大学理学部 情報科学科 馬谷研究室

Typed Racket 用 QuickCheck ライブラリ

矢作 晃 202003534

1 はじめに

QuickCheck[1] は静的型付け言語である Haskell で実装されたプログラムのテスト用ライブラリである．同じ静的型付け言語である Typed Racket への移植を行う，本要旨では，2 節で QuickCheck が行うテストの概要についてサンプルプログラムのコードと実行例を用いて紹介し，3 節では Typed Racket への移植の前段階として Racket へ移植された QuickCheck について，Haskell 版との違いを交えて紹介する．4 節では，実際に Typed Racket への移植を行い，Haskell 版におけるコード記述との差異を確認し，5 節で Haskell 版と類似したコード記述を可能とするためのマクロを実装する．最後に 6 節で本研究の総括を行う．

2 QuickCheck

QuickCheck[1] は Haskell 用のテストツールであり，ソースコード中にプログラムの性質 (property) を記述することで大量のテストを行うことができる．ここでの性質は単純な場合では，結果がブール値になる関数として定義できる．以下に文献 [2] より抜粋した加法の結合性の性質についてのサンプルプログラムを示す．

```
prop_PlusAssociative ::
    Integer -> Integer -> Integer -> Bool
prop_PlusAssociative x y z =
    (x + y) + z == x + (y + z)
```

QuickCheck においてのテストでは，この性質を記述した関数をランダムに自動生成された大量の引数で呼び出すことでテストを行う．この引数は Haskell の型推論機能によって QuickCheck が定義している生成器が呼び出されることにより生成されたものである．以下に示すのは上記のサンプルプログラムの実行例である．

```
> quickCheck prop_PlusAssociative
+++ OK, passed 100 tests.
```

また，サンプルプログラムの型シグネチャを `Integer` から `Float` に変更したときの実行例は以下のようになり，浮動小数点数の加法は結合性がないためテストに失敗していることがわかる．

```
> quickCheck prop_PlusAssociative
*** Failed! Falsified
    (after 2 tests and 7 shrinks):
-0.1
0.3
-8.0e-2
```

このように，テストに失敗した場合は，そこまでで成功したテストの数と失敗時の引数の値が表示される．

3 Racket における QuickCheck

続いて，Typed Racket への移植の前段階として，Racket へ移植された QuickCheck[3] について見ていく．

```
(require quickcheck)
(define prop_PlusAssociative
  (property ([x arbitrary-integer]
             [y arbitrary-integer]
             [z arbitrary-integer])
            (= (+ (+ x y) z)
               (+ x (+ y z)))))
```

上に示したのは，2 節と同様，加法の結合性の性質について記述したソースコードである．Racket は Haskell と異なり，動的型付け言語であり，型が存在しない．そのため，性質を記述した関数の内部で `arbitrary-integer` という整数型の引数を生成するための生成器を呼び出すことにより引数の型を決定している．

4 Typed Racket への移植

Typed Racket は Racket と同様の文法で記述ができるが，型システムを持つため，使用する関数の型情報を書き込む必要がある．これは，`require` したソースコード内で記述されている関数についても同様である．以下は実際に移植した QuickCheck のソースコードの一部である．

```
(require/typed "quickcheck/main.rkt"
  [quickcheck
    ((U property Boolean) -> Void)]
  [#:struct property
   ([proc :
    (-> Number Number Number Boolean)]
    [arg-names : (Listof Symbol)]
    [args : (Listof arbitrary)])]
  ...
  [arbitrary-integer arbitrary]
  ...
  )

(define-syntax make-property
  (syntax-rules ()
    [(make-property ((?id ?gen) ...)
                    ?body0 ?body1 ...)
     (property
      (lambda ([?id : Number] ...)
        (cast ?body0 Boolean) ?body1 ...)
      '(?id ...)
      (list (cast ?gen arbitrary) ...))]
      ))
```

この移植では，性質を記述する property 関数の引数を可変長で移植することは困難だったため，3 引数の関数の性質記述のみを可能としている．また，マクロを require することもできなかったため，コード内で再定義することにより，この問題を解決している．

加法の結合性の性質について示した関数は以下のようになり，1 行目に型情報が埋め込まれている点が Racket でのソースコードの記述と異なる．

```
(: prop_test property)
(define prop_PlusAssociative
  (make-property ([x arbitrary-integer]
                  [y arbitrary-integer]
                  [z arbitrary-integer])
                 (= (+ (+ x y) z)
                    (+ x (+ y z)))))
```

5 簡易仕様記述の追加

本研究で作成した Typed Racket 版 QuickCheck と Haskell 版 QuickCheck の大きな違いのひとつとして挙げられるのは，関数内で使用される生成器の記述の有無である．これは，Typed Racket は Haskell のような型推論機能を有していないからである．そこで，以下に示すマクロを導入することで，Typed Racket におけるラムダ式記法と似た記述で，必要な生成器を自動で呼び出すことを可能にする．

```
(define-syntax (prop-define stx)
  (syntax-case stx ()
    [(_ ([id0 : type0] ...) (body0 ...))
       #'(make-property
          ([id0 (change-arb type0)] ...)
              (body0 ...))]))

(define-syntax (change-arb stx)
  (syntax-case stx ()
    [(_ type)
     (with-syntax
        ([arb-name0 (format-id
            #'a "arbitrary-~a" #'type)])
       #'(cast arb-name0 arbitrary))]))
```

このマクロを使うと 4 節で紹介したコードは以下のように書き換えることができることを実行例と共に示す．

```
(define prop_PlusAssociative
  (prop-define ([x : integer]
                [y : integer]
                [z : integer])
               (= (+ (+ x y) z)
                  (+ x (+ y z)))))

> (quickcheck prop_PlusAssociative)
OK, passed 100 tests.
```

6 まとめ

本研究では, QuickCheck ライブラリの Typed Racket への移植を行い，Haskell 版と同様に，引数の型情報から必要な生成器を呼び出せるようにした．ただし，今回の移植でテスト可能となった関数は 3 引数関数のみであるため，これを可変長にすることは今後の課題である．

参考文献

[1] Koen Claessen and John Hughes, *QuickCheck: a lightweight tool for random testing of Haskell programs,* ICFP '00: Proceedings of the fifth ACM SIGPLAN international conference on Functional programming, pp. 268-279, 2000.

[2] ギボンズ, J. & ムーア, O. D. 編 『関数プログラミングの楽しみ』 山下伸夫訳 2010 オーム社

[3] Quickcheck by Mike Sperber and Ismael Figueroa https://docs.racket-lang.org/quickcheck/index.html