神奈川大学理学部 情報科学科 馬谷研究室

多相型と部分型を活かした Typed Racket 用 QuickCheck

岡崎 唯人 202103263

1 概要

本研究では QuickCheck[3] の静的型付き言語である Typed Racket への移植を行う．QuickCheck は Typed Racket と同様に静的型付き言語の 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．これを移植するにあたって，Racket に存在する QuickCheck ライブラリ [1] ではもとの Haskell のライブラリとは異なる部分がいくつか見受けられた．これらの点は静的型付き言語である Typed Racket に移植する際には型の安全性と型の制限という観点で無視できないものであった．先行研究では Racket で実装されたコードを Typed Racket 側から呼び出すことにより実装されていた [2] が本研究では多相型と部分型という 2 つの型を活かして型の安全性を保ちつつ，型の制限をすることとした．

2 QuickCheck

まず, 移植についての説明の前に Haskell における QuickCheck および Racket における QuickCheck についての説明を行う．前述のとおり QuickCheck[3] とは Haskell で実装されたテストツールである. 直接ソースコード中に性質を書けるようになっており，テストデータ生成言語も定義されているため，大量のテストをわずかな記述で行える．以下に例として分配法則のプログラムと QuickCheck でのテスト結果を示す．

```
prop_Distributive ::
  Integer -> Integer -> Integer -> Bool
prop_Distributive
  a b c = a * (b + c) == a * b + a * c
ghci> quickCheck prop_Distributive
+++ OK, passed 100 tests.
```

上記のプログラムでは a，b，c3 つの引数を受け取り a*(b+c)=a*b+a*c という式が成り立つかどうかを判定する関数に quickCheck を適用することでランダムに生成された任意の引数を用いて呼び出すことでテストしている．結果として 100 回呼び出したところすべて成功した．次に Racket における QuickCheck だが，上記のプログラムは以下のように実装できる．

```
(define prop_Distributive
  (property ([a arbitrary-integer]
             [b arbitrary-integer]
             [c arbitrary-integer])
            (= (* a (+ b c))
               (+ (* a b) (* a c)))))
quickcheck1.rkt>
(quickcheck prop_Distributive)
OK, passed 100 tests.
```

これらの違いとして最も大きなものは動的型付き言語である Racket には型が存在しないために，arbitrary-integer のような引数を生成するための生成器を明示的に呼び出す必要があるという点である．また，コンパイル時には型の整合性は確認されず，プログラム実行時にしか確認されないがために型の安全性も低くなっている．これは静的型付き言語と動的型付き言語の差によるものであるが，静的型付き言語の Typed Racket に移植する際には可能な限り型安全性が高まるようなものにしたい．続いて型が存在しないが故のもう 1 つの問題点が本来同じ型を使うべき場所であってもプログラム上は同じ型とは限らなくなってしまうということである．こちらも静的型付き言語であれば避けることが可能なため，Typed Racket での実装時には解消することとする．

3 Typed Racket における QuickCheck

本節では本題である Typed Racket への QuickCheck の移植について説明する．Typed Racket では Racket と同様の文法で記述でき，型推論もあるものの Haskell のような正確な型チェックを行うためにはそれぞれの関数に対して明示的な型記述が必要となる．そのため今回は Racket の QuickCheck ライブラリに型記述を行う形で Typed Racket へ移植することにした．その際に 2 節で述べた 2 点を考慮して行う．まず 1 点目のコンパイル時に型安全性が Racket での実装時よりも確認でき

るようにしたいというものである．例として Racket の QuickCheck ライブラリから１つの型をあげる．

```
;; generator   : (generator a)
;; transformer :
;;   a (generator b) -> (generator b)
(struct arbitrary
  (generator transformer))
```

この型では arbitrary は generator と transformer という 2 つのフィールドを持ち，generator は (generator a) という型，transformer は a と (generator b) を受け取り，(generator b) を返す型であるということを表している. この際, b は a と同じ型かもしれないし違う型かもしれない．しかし Racket では a と b といったような型の区別はできないため，Typed Racket ではこれらを区別しつつも同じ型でありうるような型記述を行えれば型安全性を高めることができる．そこで本研究では以下のように型付けを行った．

```
(struct (A B) Arbitrary
  ([generator : (Generator A)]
  [transformer : (-> A (Generator B)
                  (Generator B))]))
```

上記のように任意の型 A，B のような型変数を用いた多相型で実装することで型安全性を高めることができる．Typed Racket における任意の型を表現するための実装方法はもう１つあり，それはすべての型を受け入れる Any 型を使うというものである．しかし Any 型はすべての型を受け入れるがゆえに，型安全性を保てないという元の問題が解決しないために型変数を用いて実装した．

また，もう１つの同じ型を使うべき状況でも同じ型とは限らなくなってしまうという問題点を考えるためにこちらでも QuickCheck ライブラリから 1 つの例をあげる．

```
;; ok             : () = unknown, #t, #f
;; arguments-list : (list (list
;;     (pair (union #f symbol) value)))
(define-struct result
   (ok stamp arguments-list))
```

上記の result という型のフィールドのうち，arguments-list の中には value という多相型にするべきものが存在する．ここに格納されるはずの値は QuickCheck のテストで使用された値である．すなわち生成器が生成しうるような型でなければこの value の型には入れられるべきではない．そのため今までのように型変数 A を用いるのではなく，生成器の型である Generator の部分型としての型を作成し，その部分型を value の型とすることで生成器が生成しうる型のみに制限することができる．最終的に，部分型を用いて以下のように定義する事にする．

```
(struct (A) Generator ([proc :
  (-> Integer Random-generator A)]))
(define-type Gen-set
  (U Integer Char String Boolean Symbol))
(define-type Gen2
  (U (Generator Gen-set)
  (Generator (Listof Gen-set))))
```

これら２つの型の実装方法を活用することで Racket で実装されていた QuickCheck よりも Haskell のように静的型付き言語であることを活かした実装とすることができる．

4 まとめ

本研究では Typed Racket において多相型，部分型の２つを活かすことで Haskell のように型安全性が高まったより効果的な QuickCheck を実装することができた．今回はユーザーが新しく生成器と型を作成した際にその型も Result の型に入るようにすること，およびもとからパッケージに含まれていたいくつかの使用頻度の高くなさそうな生成器の型付けの 2 点ができなかったためこれらを解決する事が今後の課題である．

参考文献

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

[2] 矢作晃: 学士論文 Typed Racket 用 QuickCheck ライブラリ，令和 5 年度神奈川大学情報科学科卒研要旨集，pp. 63-64.

[3] Koen Claessen and John Hughes, QuickCheck: a lightweight tool for random testing of Haskell programs, In *Internatiional Conference on Functional Programming*, pp. 268-279, ACM, 2000