6G-03

ランダムテストを応用した プログラミング演習における答案の自動正誤判定機構

大久保　弘崇 †　　山本　晋一郎 ‡
†‡ 愛知県立大学情報科学部

1　はじめに

プログラミングの学習において、演習は重要である。その際、指導者による正誤判定、誤答に対するアドバイスなどのレスポンスが学習効果を高める。我々は、プログラミング演習問題に対する自動正誤判定機構を構築し、このレスポンスの一部を自動化した。

我々は平成25年度より大学1年向けの初等プログラミング教育にHaskell言語を採用し、正誤判定機構を用いた演習を実施してきた。本稿ではその2年の運用で得た知見をまとめる。

2　基盤技術

本研究で利用した基盤技術を紹介する。小テストを実施する環境としてMoodleを用い、プログラミング演習に対応するためにCodeRunnerを追加した。さらに、正誤判定を行うにあたり固定的なテストケースでは問題があったためランダムテストの環境としてQuickCheckを利用している。

CodeRunner　CodeRunnerはMoodleの小テストの機能を拡張する外部モジュールである。University of Canterburyの Richard Lobb らが開発しており、`http://coderunner.org.nz/` で公開されている。CodeRunnerは答案としてプログラムを入力させる。これをコンパイルし、出題者の準備したテストケースに対して実行し、正しさを自動判定する。学習者の理解を助けるため、テスト入力、正解出力、実行結果を対比して提示できる。

QuickCheck　QuickCheck[1] (以下QC) はテストケースを生成してソフトウェアテストを行うHaskell用ライブラリである。プログラマは関数が満たすべき論理的性質を記述する。これに対してQCは関数の引数の型が取り得る値をランダムに生成し関数に与え、全てが性質を満たすかを自動的に確認する。

関数に与える引数は型ごとに定義された値の生成器が適切に使用される。一般的なデータ型だけでなく、関数型についても生成器が提供されている。プログラム内で独自に定義した型に対しては、その生成器の作成を支援する機能も提供する。

3　提案手法

前節で紹介した要素技術を組み合わせることで、答案の正誤判定を機械化できる。しかし、QCはプログラマを対象としており、プログラマは良心に基づき自分のプログラムの正しさを示そうとする。演習問題に取り組む学生は時として、正しい解答を考えだす代わりに、自動判定機構を騙して正解の評価を得ることに腐心する。また、経験のあるプログラマには十分な情報を含むQCのメッセージも、初学者にとっては難解である。この間を埋めるために必要な機能を集約したテンプレートや、漏れのない性質を定義するために注意するべき点を以下で述べる。

ランダムテストの必要性　Haskellではパターンマッチにより、特定の値に対する結果を持つ関数を容易に定義できる。例えば、『与えられた整数 n に対して、1から n までの和を計算する関数 `sum1toN :: Int -> Int` を作れ』という問題において、固定的なテストケース2, 5の結果のみで正誤判定をするとする。この出題に対して、以下の答案

```
sum1toN 2 =  3
sum1toN 5 = 15
```

は正解と判定されてしまう。テストケースの開示は学習目的には必要と考えられるため、これを隠蔽できないとすると、固定的でないテストが必要となる。一方、QCの性質

```
proc :: Int -> Int
proc n = n >= 1 ==> sum1toN n == sum [1..n]
```

は、上の答案を十分な確率で誤りと判定できる。

テンプレート　提案手法による自動正誤判定の性質を記述するテンプレートをリスト1に示す。ここで、問題の要求は関数 `challenge` を定義することであり、行1の `expected` はその正解である。この名前は衝突したり、推測されにくいものにする必要がある。行3からの性質 `prop` は、行4–5で生成器や計算により引数を作

On automatic scoring of programming exercises with random test framework
†Hirotaka Ohkubo ‡Shinichiro Yamamoto
†‡Aichi Prefectural University

リスト 1: テンプレート

```
 1 expected x y = ...                    -- 正解
 2
 3 prop = do                          -- 正誤判定の性質
 4   x <- ...                           -- 引数の生成
 5   y <- ...
 6   let real = expected x y   -- 正解の計算
 7   let ans  = challenge x y -- 答案による
 8 -- エラー時のメッセージ
 9   let errmsg = unwords ["challenge",
10      show x, show y, "is", show real,
11      ",␣but␣returned", show ans]
12 -- 比較の実行
13   return $ whenFail (putStrLn errmsg)
14     $ real == ans
```

り、行 6–7 で引数に対する expected と challenge の結果を求め、行 13–14 で両者が等しいか比較している。等しくない場合、行 9–11 で用意したエラーメッセージを表示する。

固定的なテストとランダムテストの併用　ランダムテストは強力な道具であるが、全てのテストに取って代わるものではない。問題が明確なコーナーケースを含む場合、ランダムテストだけではその点を網羅できないため、固定的なテストケースとして別に確認するべきである。また、問題文に実例として示したケースを固定テストとして含めることで、その実行結果を学習者に提示できる。

正解関数を用いない性質記述　一般的な QC の利用場面では、作成中のプログラムが満たすべき性質を記述する。これに対して、演習問題には出題者が想定する正解が存在するため、テンプレートは正解関数と答案の結果を比較する構成になっている。しかし万一この正解関数にバグが含まれていた場合、バグのない本当に正しい答案を不正解と誤判定する。これを防ぐため、可能ならば一般的な QC の利用法と同様に、プログラムの性質を用いた記述をすることが望ましい。

多相性の考慮　QC は型に基づき最適な生成器を選択するが、時として最適すぎて妥当でないものを使用する場合がある。この問題への対応のため、正誤判定に QC を応用する場合には、全ての引数の型を固定的に与えることが望ましい。

しかし一方で、演習問題として多相性のある定義を要求したい場合がある。例えば、`elem :: Eq a => a -> [a] -> Bool` 関数の再実装を

```
elem :: Int -> [Int] -> Bool
elem _ [] = False
elem a (x:xs) = (a - x == 0) || elem a xs
```

のようにすると、Int 型については正しく振る舞う。そのため、性質が Int 型でしか確認しないと、正解と誤判定される。

この問題は、固定的なテストケースにおいて、Eq クラスの別のインスタンスを用いた例を確認することで回避できる。

学習態度の問題　Moodle の提供する Web ベースの対話環境により、学生はターンアラウンドの速い自習が可能になる。しかし、そのターンアラウンドの速さゆえに、学生は Moodle の画面でプログラミングをし、正しさの確認は自動判定に任せるという利用形態が一般化した。従来のプログラミングの自習では、コーディング後のテスト実行において、作成中のプログラムがどのような入力を与えたらどのような結果を返すはずか、といったプログラムの振る舞いを想像し、テストケースを自ら作成する必要がある。上述の利用形態を可能にする提案手法は、初学者からテストケース作成の訓練をする機会を奪っているともいえる。

Moodle の小テストモジュールには、判定を行うたびに問題の総得点を割り引く機能がある。必要ならばこれを適用することで、むやみな自動判定の起動を抑制できると考えられる。

例外の扱い　整数の 0 除算や、網羅的でないパターンマッチを評価した場合に実行時例外が発生する。QC はテストの失敗と実行時例外を区別し、後者で直ちにテストの実行を放棄する。QC のこの振る舞いは、テスト対象関数が正しい入力範囲において正しい結果を返すことは検査できるが、正しくない入力に対して例外を発生させる動作が仕様であったとしてもそれを確認できない。例外を含めた振る舞いを正解関数と突き合わせられるような提案手法の改良は今後の課題である。

4　発展

QC は、引数の数によらず、性質の型に基づき妥当な生成器を適用する機能を持つ。同様の手法を用いることで、現在はテンプレートの編集で実現している機構を任意個任意の型の引数の関数を統一的にテストできる一つの関数に抽象化できればより利用しやすくなる。これ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. In *International Conference on Functional Programming*. ACM, 2000.