ユニットテストあれこれ

～ Haskeller の視点から ～

2018.3.9

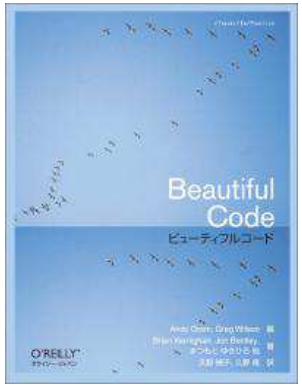
山本和彦

おわび

時間がなかったので
例題に一貫性がありません

おしながき

- ビューティフル・テストより
  - スモークテスト
  - 境界テスト
  - ランダムテスト
  - 突然変異テスト
- Haskellerの視点より
  - スモークテストとドキュメント
  - 性質テスト
  - 性質テストと突然変異テスト



「ビューティフルコード」
7章 ビューティフル・テスト

二分探索に対するテスト

二分探索のアイディアは1946年に出された
バグがない実装ができたのは12年後

例：二分探索

- `binarySearch` の仕様
  - 要素を発見したら、配列内の位置(何番目か)を返す
  - 要素が発見できなかったら、`-1` を返す
- Java っぽい文法を使います

(1) スモークテスト

- 単純な利用例を使ったテスト
- 語源
  - ハードが完成したら、まず電源を入れて
    煙が出ないか確かめたことから
- 二分探索での例
  - arr = { 1, 4, 42, 55, 67, 87, 100, 245 }
  - `assertEquals( 2, binarySearch(arr, 42))`
  - `assertEquals(-1, binarySearch(arr, 43))`

(2) 境界テスト

- 配列の大きさの境界
  - `assertEquals(-1, binarySearch({    }, 42))`
  - `assertEquals( 0, binarySearch({ 42 }, 42))`
  - `assertEquals(-1, binarySearch({ 42 }, 43))`
  - 大きい方の境界も試すべきだが難しい
- 要素の位置の境界
  - `arr = { -324, -3, -1, 0, 42, 99, 101 }`
  - `assertEquals(0, binarySearch(arr, -324)) // 左端`
  - `assertEquals(3, binarySearch(arr,    0)) // 真ん中`
  - `assertEquals(6, binarySearch(arr,  101)) // 右端`
- 人間が考えるのは馬鹿らしい
  - 自動化したい

(3) ランダムテスト

- 入力を自動生成してテストする
  - 入力はランダムに生成する
  - 入力の生成方法がよければコーナーケースを発見できる
- コードが持つべき性質
  - `binarySearch` が `-1` を返すとき、要素は配列に含まれていない
  - `binarySearch` が `0` 以上を返すとき、配列のその位置に要素がある
- 性質はこれだけで十分か？

(4) 突然変異テスト (1)

- 突然変異したコード
  - mutant

```
binarySearchX(int[] arr, int target) {
  r = binarySearch(arr, target);
  if (r != -1) {
    return r;
  } else {
    if (target <= 424242) {
      return -1;
    else {
      return -42;
    }
}
```

- 突然変異の特徴
  - 十分な性質を書いておけば、突然変異はテストをすり抜けられない
  - 突然変異がテストをすり抜けたとしたら、テストは十分ではない

(4) 突然変異テスト (2)

- これまで、出力がこうなら入力はこうだと考えてきた
  - `binarySearch` が `-1` を返すとき、要素は配列に含まれていない
  - `binarySearch` が `0` 以上を返すとき、配列のその位置に要素がある
- 突然変異したコード(再掲)
  - `target` が `424242` 以下ならテストをすり抜ける

```
binarySearchX(int[] arr, int target) {
  r = binarySearch(arr, target);
  if (r != -1) {
    return r;
  } else {
    if (target <= 424242) {
      return -1;
    else {
      return -42;
    }
}
```

(4) 突然変異テスト (3)

- 入力がこうなら出力はこうだというテストも必要
  - 要素が配列のn番目に含まれているときは、
    `binarySearch` は `n` を返す
  - 要素が配列に含まれていないときは、`binarySearch` は `-1` を返す
- 突然変異したコード(再掲)
  - `target` が `424242` 以下でもテストをすり抜けられない

```
binarySearchX(int[] arr, int target) {
  r = binarySearch(arr, target);
  if (r != -1) {
    return r;
  } else {
    if (target <= 424242) {
      return -1;
    else {
      return -42;
    }
}
```

Haskellerからの視点

スモークテストとドキュメント

- 利用例はマニュアルに書くべき
  - テストに書いておくのはもったいない
- Haskell では Python から doctest を輸入
  - ドキュメントに書いてる例が実際その通りになるかテストする

Python の doctest

- docstring に使用例を書く

```
def factorial(n):
  """Return the factorial of n,
  an exact integer >= 0.
  If the result is small enough to fit in an int,
  return an int. Else return a long.

  >>> [factorial(n) for n in range(6)]
  [1, 1, 2, 6, 24, 120]
  >>> factorial(30)
  265252859812191058636308480000000L
  """
  import math
  if not n >= 0:
    raise ValueError("n must be >= 0")
   ...
```

- 本家の Python ではあまり使われてないようだ
  - 動的言語なので気軽に返り値を変えてしまう傾向がある？

Haskell の doctest

- 対話環境で動かした結果をマニュアルにコピペする

```
-- |
-- Base64 encoding.
--
-- >>> encode "foo bar"
-- "Zm9vIGJhcg=="
encode :: String -> String
encode = ...
```

- マニュアルの生成

```
encode :: String -> String                                  # Source
```

Base64 encoding.

```
>>> encode "foo bar"
"Zm9vIGJhcg=="
```

- 自動テスト

性質テスト

- コードのテストとは？
  - コードが持つ性質をテストすること ← ここ大事
  - 証明も同じ。コード自体の証明ではなく、性質を証明する
- 性質テスト
  - property test
  - ランダムテストより一歩考え方が進んでいる
  - 入力はランダムに生成されるとは限らない
  - 入力を自動的に生成しながら、性質をテストする
  - コーナーケースを自動的に発見できる

Haskell での性質テスト

- QuickCheck
  - 最も有名な性質テストツール
  - Haskell で発祥
  - 多くの言語に移植されている
  - テストデータは乱数的に生成。ただし大きさを次第に大きくする
  - doctest にも書ける！
  - それぞれのデータ型が `arbitrary` インターフェイスを実装
  - TLS ライブラリも QuickCheck でテストしている
- SmallCheck
  - Haskell 由来の性質テストツール
  - テストデータは、ある深さに対し、全てを網羅するよう生成

性質テストの例(1)

- 行ったら戻れるかのテスト
  - `decode(encode(inp)) == inp`
  - `deserialize(serialize(inp)) == inp`
  - `parse(prittyPrint(inp)) == inp`

性質テストの例(2)

- ビューティフル・テストを思い出そう
  - スモークテスト
  - 境界テスト
  - ランダムテスト
  - 突然変異テスト
- これは単に線形探索と振る舞いが同じというだけでは？
  - 計算量を除いて
- モデル実装を使ったテスト
  - `linearSearch(arr, x) == binarySearch(arr, x)`

性質テストと突然変異テスト

- 性質テストに関する不安
  - 性質が不足していないか？
    - 十分なテストではない
  - 性質が重複していないか？
    - リソースの無駄遣い
- 突然変異を自動生成する
  - 突然変異がすべての性質をすり抜けたら性質が不足している
  - たくさんの突然変異を作りそれぞれの性質を試す
    - すり抜け/捕獲の数が同じなら、同じ性質
    - 突然変異が性質Aをすり抜けるときに必ず性質Bもすり抜けるなら、
      性質Aから性質Bを導ける
- これを実現するのが FitSpec

FitSpec (1)

- リストの要素をソートする例
  - 性質1：ソートの後では要素は小さい順に並んでいる
  - 性質2：ソートの前と後で、リストの大きさは変わらない
  - 性質3：ある値がリストの要素なら、ソート後も要素である
  - 性質4：ある値がリストの要素でないなら、ソート後も要素でない
  - 性質5：ソート前のリスト中の最小値は、ソート後は先頭にある
- FitSpec にかけると
  - 3つの突然変異がすり抜ける
    - 最小例は `sort({0, 0, 1})` → `{0, 1, 1}`
  - 性質3と性質4は同じ。性質1と性質3から性質5が導ける
- 性質の改善
  - 性質1：ソートの後では要素は小さい順に並んでいる
  - 性質2：ソートの前と後で、リストの大きさは変わらない
  - 性質3：ある値がリストの要素なら、ソート後も要素である
  - 性質6：ある値がリストに含まれている個数は、ソート後も同じ

FitSpec (2)

- 性質の改善 (再掲)
  - 性質1：ソートの後では要素は小さい順に並んでいる
  - 性質2：ソートの前と後で、リストの大きさは変わらない
  - 性質3：ある値がリストの要素なら、ソート後も要素である
  - 性質6：ある値がリストに含まれている個数は、ソート後も同じ
- FitSpec にかけると
  - 突然変異のすり抜けなし
  - 性質6から性質2と性質3が導ける
- 最終的な性質 (完全かつ重複なし)
  - 性質1：ソートの後では要素は小さい順に並んでいる
  - 性質6：ある値がリストに含まれている個数は、ソート後も同じ