PHPMatsuri Reboot in Hackone

成果発表LT



pixiv Inc.
USAMI Kenta

pixiv

先週新潟でプロポーザル書いてたんですよ


にゃんだーすわん
@tadsan
Show translation
あたしたっどさん、新潟駅前でハロウィーンに興じる若者たちを見守りながらプロポーザルを書いてるの #phpkansai
10:45 PM · Oct 31, 2026 · 2,813 Views

pixiv

PHPカンファレンス関西2026

レギュラートーク（20分）

Property-based Testingを越えて



うさみけんた　　tadsan

☆ 5

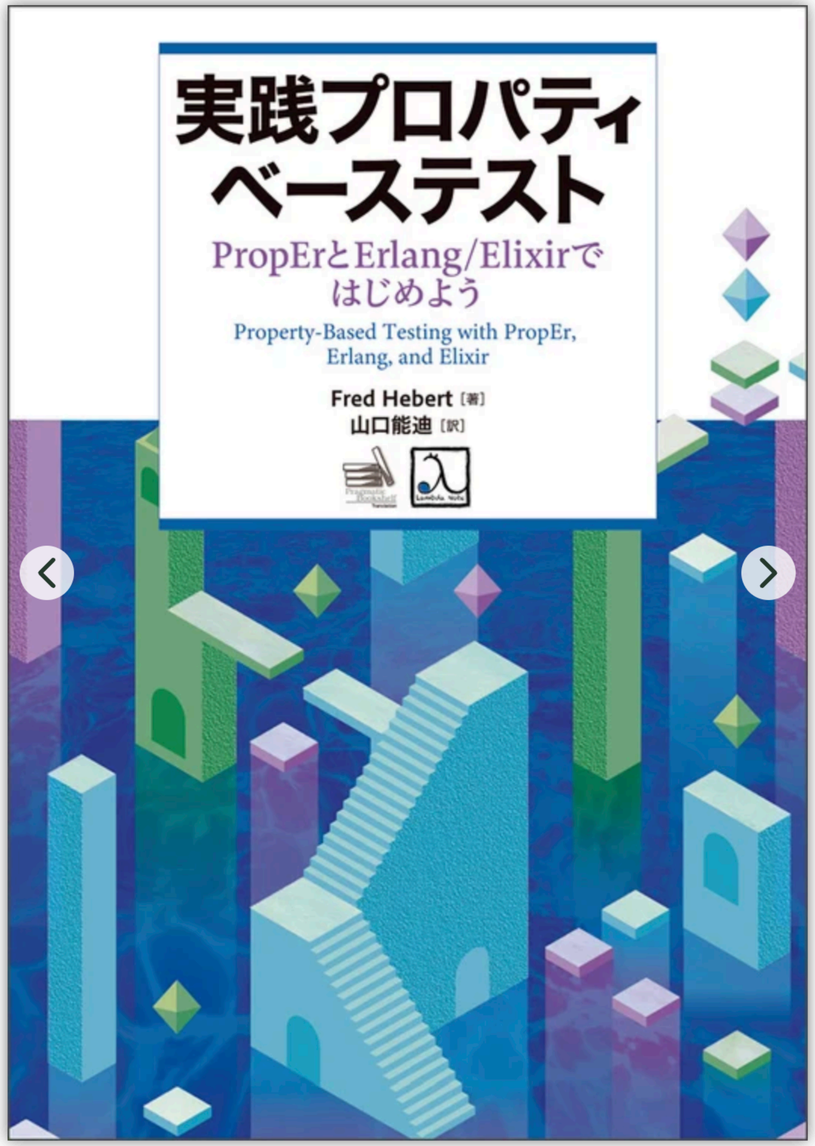
あなたが書いた関数はきちんと動きますか？ どんな入力を与えても動きますか？

HaskellのQuickcheckを嚆矢とするProperty-based Testing(PBT)と呼ばれる手法は型情報をもとに自動テストを行なうことができます。PHPでのPBTには既にいくつも実装がありますが、PHPのAttributeを応用したDSLを設計することでオリジナルのQuickcheckよりもさらに宣言的なテストを実現できます。

本トークではPBTの基本的なアイディアと、そのエレガントな手法がどのくらい役に立ったり、それほど役に立たなかったりするのかについて紹介します。

pixiv

PBTというものがある



実践プロパティベーステスト — PropErとErlang/Elixirではじめよう

¥4,500（税込¥4,950）

カートに追加

- 紙書籍をお届けします（PDFがついてきます）
  - PDFのみ必要な場合は、こちらからPDF単体を購入できます
- 通常はご注文から2〜3営業日で発送します
- 年末年始や大型連休など、1週間から10日程度、配送のお休みをいただく場合があります。詳しくはお知らせをご覧ください

テストケースはコンピューターで書くべき！ でもどうやって？ その答えが「プロパティベーステスト」です

- Fred Hebert 著、山口能迪 訳
- 原書 Property-Based Testing with PropEr, Erlang, and Elixir
- 376ページ
- A5判/1色刷
- ISBN：978-4-908686-18-4
- 2023年11月1日 発行

pixiv

めちゃくちゃいい本

ちょっと厚い

この本はErlang/Elixir
という
言語で書かれている

元ねたはHaskellのQuickCheck

型とか満たすべき性質を宣言するとランダムな値でテストしてくれる

意外な値でテストが
失敗したりして
穴が見つかると嬉しい

PHPでもやりたい

Eris



Eris is a porting of QuickCheck and property-based testing tools to the PHP and PHPUnit ecosystem.

In property-based testing, several properties that the System Under Test must respect are defined, and a large sample of generated inputs is sent to it in an attempt to break the properties.

Compatibility

- PHP 8.1, 8.2, 8.3, 8.4, 8.5
- PHPUnit 10.x, 11.x, 12.x, 13.x

Installation

You can install Eris through Composer by running the following command in your terminal:

```
composer require --dev giorgiosironi/eris
```

You can run some of Eris example tests with `vendor/bin/phpunit vendor/giorgiosironi/eris/examples`.

Here is an empty sample project installing Eris.

pixiv

あるんだけど

Example usage within PHPUnit

This test tries to verify that natural numbers from 0 to 1000 are all smaller than 42. It's a failing test designed to show you an example of error message.

```php
<?php
use Eris\Generators;

class ReadmeTest extends \PHPUnit\Framework\TestCase
{
    use \Eris\TestTrait;

    public function testNaturalNumbersMagnitude()
    {
        $this->forAll(
            Generators::choose(0, 1000)
        )
            ->then(function($number) {
                $this->assertTrue(
                    $number < 42,
                    "$number is not less than 42 apparently"
                );
            });
    }
}
```

Eris generates a sample of elements from the required domain (here the integers from 0 to 1000) and verifies a property on each of them, stopping at the first failure.

pixiv

型みたいなのを使うなら
型 + #[Attr]で
宣言できてよくない？

pixiv

ということで作ってた

```php
class ArraySearcher
{
    /**
     * 1. $items を先に生成し、#[Ordered] を入力生成モディファイアとして適用して昇順ソート済みにする
     * 2. $target は「80%の確率で $items の要素から選び、20%の確率で任意のintを生成する」依存Generator
     *
     * @phpstan-param non-empty-list<int<-1000, 1000>> $items
     * @phpstan-return int<-1, max>
     */
    #[Verify]
    #[Ensures('$return === -1 ? !in_array($target, $items, true) : $items[$return] === $target')]
    public static function binarySearch(
        #[Gen\Sorted]
        array $items,

        #[Gen\OneOf(
            new Gen\ElementOf('$items', weight: 80),
            new Gen\FromType(weight: 20), // PHPDoc/型宣言の int から生成
        )]
        int $target,
    ): int {
```


pixiv

```php
class ArraySearcher
{
    /**
     * 1. $items を先に生成し、#[Ordered] を入力生成モディファイアとして適用して昇順ソート済みにする
     * 2. $target は「80%の確率で $items の要素から選び、20%の確率で任意のintを生成する」依存Generator
     *
     * @phpstan-param non-empty-list<int<-1000, 1000>> $items
     * @phpstan-return int<-1, max>
     */
    #[Verify]
    #[Ensures('$return === -1 ? !in_array($target, $items, true) : $items[$return] === $target')]
    public static function binarySearch(
        #[Gen\Sorted]
        array $items,

        #[Gen\OneOf(
            new Gen\ElementOf('$items', weight: 80),
            new Gen\FromType(weight: 20), // PHPDoc/型宣言の int から生成
        )]
        int $target,
    ): int {
```


pixiv

これはPHPで既に書かれてる

```php
class ArraySearcher
{
    /**
     * 1. $items を先に生成し、#[Ordered] を入力生成モディファイアとして適用して昇順ソート済みにする
     * 2. $target は「80%の確率で $items の要素から選び、20%の確率で任意のintを生成する」依存Generator
     *
     * @phpstan-param non-empty-list<int<-1000, 1000>> $items
     * @phpstan-return int<-1, max>
     */
    #[Verify]
    #[Ensures('$return === -1 ? !in_array($target, $items, true) : $items[$return] === $target')]
    public static function binarySearch(
        #[Gen\Sorted]
        array $items,

        #[Gen\OneOf(
            new Gen\ElementOf('$items', weight: 80),
            new Gen\FromType(weight: 20), // PHPDoc/型宣言の int から生成
        )]
        int $target,
    ): int {
```


pixiv

これはPHPで既に書かれてる

```php
class ArraySearcher
{
    /**
     * 1. $items を先に生成し、#[Ordered] を入力生成モディファイアとして適用して昇順ソート済みにする
     * 2. $target は「80%の確率で $items の要素から選び、20%の確率で任意のintを生成する」依存Generator
     *
     * @phpstan-param non-empty-list<int<-1000, 1000>> $items
     * @phpstan-return int<-1, max>
     */
    #[Verify]
    #[Ensures('$return === -1 ? !in_array($target, $items, true) : $items[$return] === $target')]
    public static function binarySearch(
        #[Gen\Sorted]
        array $items,

        #[Gen\OneOf(
            new Gen\ElementOf('$items', weight: 80),
            new Gen\FromType(weight: 20), // PHPDoc/型宣言の int から生成
        )]
        int $target,
    ): int {
```


pixiv

これはPHPで既に書かれてる

実装に書くと勝手にテスト

```php
class ArraySearcher
{
    /**
     * 1. $items を先に生成し、#[Ordered] を入力生成モディファイアとして適用して昇
     * 2. $target は「80%の確率で $items の要素から選び、20%の確率で任意のintを生
     *
     * @phpstan-param non-empty-list<int<-1000, 1000>> $items
     * @phpstan-return int<-1, max>
     */
    #[Verify]
    #[Ensures('$return === -1 ? !in_array($target, $items, true) : $items[$return] === $target')]
    public static function binarySearch(
        #[Gen\Sorted]
        array $items,

        #[Gen\OneOf(
            new Gen\ElementOf('$items', weight: 80),
            new Gen\FromType(weight: 20), // PHPDoc/型宣言の int から生成
        )]
        int $target,
    ): int {
```


pixiv

これはPHPで既に書かれてる

実装に書くと勝手にテスト

```php
class ArraySearcher
{
    /**
     * 1. $items を先に生成し、#[Ordered] を入力生成モディファイアとして適用して昇
     * 2. $target は「80%の確率で $items の要素から選び、20%の確率で任意のintを生
     *
     * @phpstan-param non-empty-list<int<-1000, 1000>> $items
     * @phpstan-return int<-1, max>
     */
    #[Verify]
    #[Ensures('$return === -1 ? !in_array($target, $items, true) : $items[$return] === $target')]
    public static function binarySearch(
        #[Gen\Sorted]
        array $items,

        #[Gen\OneOf(
            new Gen\ElementOf('$items', weight: 80),
            new Gen\FromType(weight: 20), // PHPDoc/型宣言の int から生成
        )]
        int $target,
    ): int {
```


pixiv

```php
class ArraySearcher
{
    /**
     * 1. $items を先に生成し、#[Ordered] を入力生成モディファイアとして適用して昇
     * 2. $target は「80%の確率で $items の要素から選び、20%の確率で任意のintを生
     *
     * @phpstan-param non-empty-list<int<-1000, 1000>> $items
     * @phpstan-return int<-1, max>
     */
    #[Verify]
    #[Ensures('$return === -1 ? !in_array($target, $items, true) : $items[$return] === $target')]
    public static function binarySearch(
        #[Gen\Sorted]
        array $items,

        #[Gen\OneOf(
            new Gen\ElementOf('$items', weight: 80),
            new Gen\FromType(weight: 20), // PHPDoc/型宣言の int から生成
        )]
        int $target,
    ): int {
```

これはPHPで既に書かれてる

実装に書くと勝手にテスト

テストの事情が漏れ出てる

pixiv

いったんよくある
PBTを満たせるように

AIがまだ常識に囚われてる気がする

俺たちの戦いは
これからだ！

ちなみに関西には
別のテーマが
採択されました