# Property Testing with derived idempotents

Brian Zeligson

April 2020

## Table of Contents

## Table of Contents

## Challenges of Property Testing

You don't know what your inputs are.

## Challenges of Property Testing

You don't know what your inputs are.

Your properties are meant to hold over a broad set of inputs, they must be general.

## Challenges of Property Testing

You don't know what your inputs are.

Your properties are meant to hold over a broad set of inputs, they must be general.

How do you make meaningful assertions without re-implementing the code under test?

## Challenges of Property Testing

You don't know what your inputs are.

Your properties are meant to hold over a broad set of inputs, they must be general.

How do you make meaningful assertions without re-implementing the code under test?

How is it handled at the source?

QuickCheck
Hypothesis
JSVerify

## Table of Contents

## Isomorphism Defined

$$\begin{tikzcd} A \arrow[r, bend left, "to"] & B \arrow[l, bend left, "from"] \end{tikzcd}$$

$$\text{from}(\text{to}(\text{A})) = \mathit{from} \circ \mathit{to} = 1_A$$
$$\text{to}(\text{from}(\text{B})) = \mathit{to} \circ \mathit{from} = 1_B$$

## Isomorphism Example: encode $\Longleftrightarrow$ decode

$$\begin{tikzcd}
JSON \arrow[r, bend left, "encode"] & String \arrow[l, bend left, "decode"]
\end{tikzcd}$$

$$\text{decode}(\text{encode}(\text{JSON})) = decode \circ encode = 1_{JSON}$$
$$\text{encode}(\text{decode}(\text{String})) = encode \circ decode = 1_{String}$$

## Isomorphism Example: reverse $\iff$ reverse

$$\begin{tikzcd}
String \arrow[r, bend left, "reverse"] & String \arrow[l, bend left, "reverse"]
\end{tikzcd}$$

$$\text{reverse}(\text{reverse}(\text{String})) = reverse \circ reverse = 1_{String}$$

$$\text{reverse}(\text{reverse}(\text{String})) = reverse \circ reverse = 1_{String}$$

## Table of Contents

## Idempotent Defined

$$\begin{tikzcd} A \arrow[r, bend left, "to"] & B \arrow[l, bend left, "from"] \end{tikzcd}$$

$$\text{to}(\text{from}(\text{B})) = to \circ from = 1_B$$

## Idempotent Defined

$$\begin{tikzcd} A \arrow[r, bend left, "to"] & B \arrow[l, bend left, "from"] \end{tikzcd}$$

$$\text{to}(\text{from}(\text{B})) = \textit{to} \circ \textit{from} = 1_B$$

$$\text{from}(\text{to}(\text{from}(\text{to}(\text{A})))) =$$

## Idempotent Defined

$$\begin{tikzcd} A \arrow[r, bend left, "to"] & B \arrow[l, bend left, "from"] \end{tikzcd}$$

$$\mathrm{to(from(B))} = \mathit{to} \circ \mathit{from} = 1_B$$

$$\mathrm{from(to(from(to(A))))} =$$
$$\mathit{from} \circ \mathit{to} \circ \mathit{from} \circ \mathit{to} =$$

## Idempotent Defined

$$\begin{tikzcd} A \arrow[r, bend left, "to"] & B \arrow[l, bend left, "from"] \end{tikzcd}$$

$$\mathrm{to}(\mathrm{from}(\mathrm{B})) = to \circ from = 1_B$$

$$\mathrm{from}(\mathrm{to}(\mathrm{from}(\mathrm{to}(\mathrm{A})))) =$$
$$from \circ to \circ from \circ to =$$
$$from \circ (to \circ from) \circ to =$$

## Idempotent Defined

$$\begin{tikzcd} A \arrow[r, bend left, "to"] & B \arrow[l, bend left, "from"] \end{tikzcd}$$

$$\text{to}(\text{from}(\text{B})) = to \circ from = 1_B$$

$$\text{from}(\text{to}(\text{from}(\text{to}(\text{A})))) =$$

$$from \circ to \circ from \circ to =$$

$$from \circ (to \circ from) \circ to =$$

$$from \circ 1_B \circ to =$$

## Idempotent Defined

$$\begin{tikzcd} A \arrow[r, bend left, "to"] & B \arrow[l, bend left, "from"] \end{tikzcd}$$

$$\text{to}(\text{from}(\text{B})) = to \circ from = 1_B$$

$$\begin{aligned}
&\text{from}(\text{to}(\text{from}(\text{to}(\text{A})))) = \\
&from \circ to \circ from \circ to = \\
&from \circ (to \circ from) \circ to = \\
&from \circ 1_B \circ to = \\
&from \circ to
\end{aligned}$$

## Idempotent Example: Run $\hookrightarrow$ Rest

$$\begin{tikzcd} Run \arrow[r, bend left, "finish"] & Rest \arrow[l, bend left, "start"] \end{tikzcd}$$

$$\text{start}(\text{finish}(\text{Run})) = start \circ finish = 1_{Run}$$

## Idempotent Example: Run $\hookrightarrow$ Rest

$$\begin{tikzcd} Run \arrow[r, bend left, "finish"] & Rest \arrow[l, bend left, "start"] \end{tikzcd}$$

$$\text{start}(\text{finish}(\text{Run})) = start \circ finish = 1_{Run}$$

$$\text{finish}(\text{start}(\text{finish}(\text{start}(\text{Rest})))) =$$
$$finish \circ start \circ finish \circ start =$$
$$finish \circ (start \circ finish) \circ start =$$
$$finish \circ (1_{Run}) \circ start =$$
$$finish \circ start$$

## Idempotent Example: JSON $\hookrightarrow$ String

$$\begin{tikzcd} JSON \arrow[r, bend left, "encode"] & String \arrow[l, bend left, "decode"] \end{tikzcd}$$

$$\text{decode}(\text{encode}(\text{JSON})) = decode \circ encode = 1_{JSON}$$

## Idempotent Example: JSON $\hookrightarrow$ String

$$\begin{tikzcd} JSON \arrow[r, bend left, "encode"] & String \arrow[l, bend left, "decode"] \end{tikzcd}$$

$$\text{decode}(\text{encode}(\text{JSON})) = decode \circ encode = 1_{JSON}$$

$$\text{encode}(\text{decode}(\text{encode}(\text{decode}(\text{String})))) =$$
$$encode \circ decode \circ encode \circ decode =$$
$$encode \circ (decode \circ encode) \circ decode =$$
$$encode \circ (1_{JSON}) \circ decode =$$
$$encode \circ decode$$

## Table of Contents

## Making Properties Easy

We know that properties are easy and effective when we have an isomorphism.

```
prop_RevRev xs = reverse (reverse xs) == xs
  where types = xs::[Int]
```

## Making Properties Easy

We know that properties are easy and effective when we have an isomorphism.

```
prop_RevRev xs = reverse (reverse xs) == xs
  where types = xs::[Int]
```

What about when we don’t have an isomorphism?

## Making Properties Easy

We know that properties are easy and effective when we have an isomorphism.

```
prop_RevRev xs = reverse (reverse xs) == xs
  where types = xs::[Int]
```

What about when we don't have an isomorphism?

Can we *find* an isomorphism?

## Finding an isomorphism

FizzBuzz does not belong to an isomorphism.

```
object FizzBuzz {
  def apply(nums: List[Int]): List[String] =
    nums.map(n => (n, n % 3, n % 5) match {
      case (0, _, _) => "0"
      case (_, 0, 0) => "FizzBuzz"
      case (_, 0, _) => "Fizz"
      case (_, _, 0) => "Buzz"
      case _ => n.toString
    })
}
```

## Finding an isomorphism

FizzBuzz does not belong to an isomorphism.

```
object FizzBuzz {
  def apply(nums: List[Int]): List[String] =
    nums.map(n => (n, n % 3, n % 5) match {
      case (0, _, _) => "0"
      case (_, 0, 0) => "FizzBuzz"
      case (_, 0, _) => "Fizz"
      case (_, _, 0) => "Buzz"
      case _ => n.toString
    })
}
```

What is the closest isomorphism we can find?

It helps to take a different perspective.

## FizzBuzz as a Set Function

$$\begin{tikzcd}
 & 1 \arrow[r] & 1 & \\
 & 2 \arrow[r] & 2 & \\
 & 3 \arrow[r] & \mathit{Fizz} & \dots \\
\dots & 5 \arrow[r] & \mathit{Buzz} & \dots \\
\dots & 9 \arrow[uur] & & \dots \\
\dots & 12 \arrow[uuur] & & \dots \\
\dots & 15 \arrow[r] & \mathit{FizzBuzz} & \dots \\
\dots & 25 \arrow[uuuur] & & \dots \\
\dots & 30 \arrow[uur] & & \dots
\end{tikzcd}$$

We cannot have an isomorphism because inputs *collapse* onto outputs.
This prevents construction of an inverse.

## FizzBuzz as a Set Function, Partitioned Domain

$$
\begin{array}{cccc}
 & \{1\} \longrightarrow 1 & \\
 & \{2\} \longrightarrow 2 & \\
 & \{3, 6, 9, 12, \ldots\} \longrightarrow \textit{Fizz} & \\
 & \{4\} \longrightarrow 4 & \\
 & \{5, 10, 20, 25, \ldots\} \longrightarrow \textit{Buzz} & \\
 & \{7\} \longrightarrow 7 & \ldots \\
\ldots & \{15, 30, \ldots\} \longrightarrow \textit{FizzBuzz} & \\
 & \{16\} \longrightarrow 16 &
\end{array}
$$

We have an isomorphism, can we fix the input type?

## FizzBuzz as a Set Function, Partitioned Domain

$$\begin{tikzcd}
\{1\} \arrow[r] & 1 \\
\{2\} \arrow[r] & 2 \\
\{3,6,9,12,\ldots\} \arrow[r] & \textit{Fizz} \\
\{4\} \arrow[r] & 4 \\
\{5,10,20,25,\ldots\} \arrow[r] & \textit{Buzz} \\
\{7\} \arrow[r] & 7 & \ldots \\
\ldots \quad \{15,30,\ldots\} \arrow[r] & \textit{FizzBuzz} \\
\{16\} \arrow[r] & 16
\end{tikzcd}$$

We have an isomorphism, can we fix the input type?

With an idempotent.

## FizzBuzz$^{-1}$ as a Set Function, Idempotent

We just pick one value from each input set.

$$\begin{tikzcd}
 & 1 & \{1\} \arrow[l] & 1 \arrow[l] & \\
 & 2 & \{2\} \arrow[l] & 2 \arrow[l] & \\
 & 3 & \{3, 6, 9, 12, \ldots\} \arrow[l] & \mathit{Fizz} \arrow[l] & \\
 & 4 & \{4\} \arrow[l] & 4 \arrow[l] & \\
 & 5 & \{5, 10, 20, 25, \ldots\} \arrow[l] & \mathit{Buzz} \arrow[l] & \\
 & 7 & \{7\} \arrow[l] & 7 \arrow[l] & \ldots \\
\ldots & 15 & \{15, 30, \ldots\} \arrow[l] & \mathit{FizzBuzz} \arrow[l] & \\
 & 16 & \{16\} \arrow[l] & 16 \arrow[l] &
\end{tikzcd}$$

This can be pre-composed with FizzBuzz to create an identity on the output set.

This means we have an idempotent on the input set.

## Table of Contents