

Property-Based Testing

How to test a program?

- Why is testing important?
- A code tester walks into a bar
 - Orders a beer
 - Orders ten beers
 - Orders 2.15 billion beers
 - Orders -1 beer
 - Orders a nothing
 - Orders a lizard
 - Tries to leave without paying

Let's test rev (list reverse)... with a unit test

```
let rec rev l =  
  match l with  
  | [] → []  
  | h :: t → rev t @ [h]
```

```
let test_reverse =  
  reverse [1;2;3] = [3;2;1]
```

function
under test

sample
argument

expected
result

Unit Testing

- What is Unit Testing?
- Disadvantages
 - Hard coded tests
 - Difficult to write good unit tests
 - Time consuming
 - Have to write many tests
 - Repeated/redundant tests

Properties

- Instead of unit tests on specific inputs and outputs, what if we could test properties that hold for *all* inputs?

```
let prop_reverse l = rev (rev l) = l
```

- reversing a list twice gives back the original list

Property Based Testing

- **Property testing consists of:**

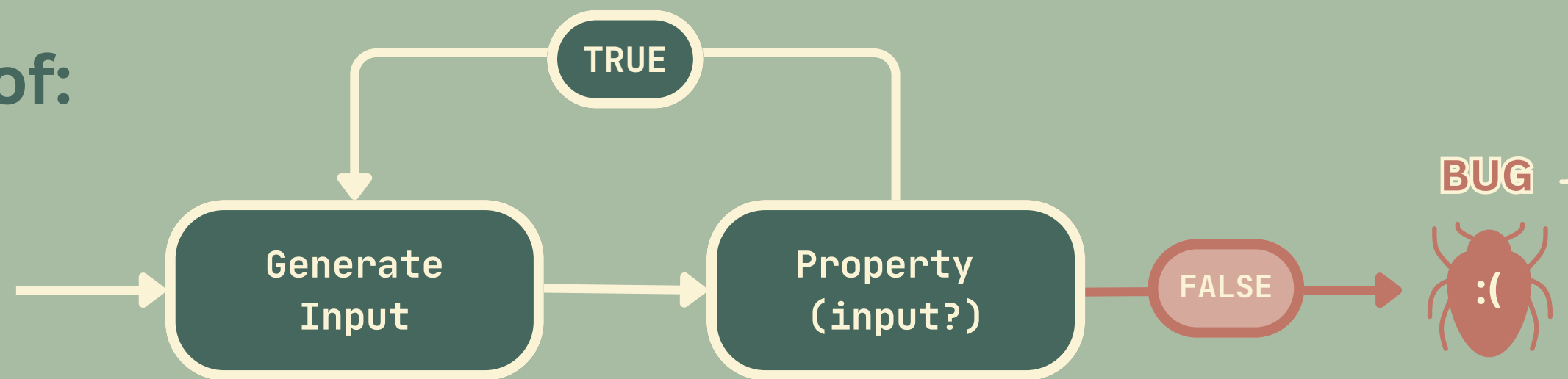
- Random input that is tested on a property

- **What is a property??**

- A feature that applies to outputs of all valid implementation of the function
- Usually represented as a function that outputs a boolean!

- **Advantages**

- Speed up testing
- Help catch weird edge cases that are hard to come up with



Example

- **Fall '24 Quiz 2**

`signed_square` should take in an `int x`, square it, but keep the original sign

- **Example properties - some aren't valid!**

- The output of `signed_square` should be greater than or equal to the input
- The output of `signed_square x` should have the same sign as `x`

```
let signed_square x = if x < 0 then ( - x ) * ( - x ) else x * x
```

Good/Bad Properties

- Good or bad? Why?

- *The output of `signed_square` should be greater than or equal to the input*

- Bad :(

- some inputs of a valid implementation $\rightarrow$ returns false

- Ex. `signed_square -4` $\rightarrow$ returns false because $-16 < -4$

- *The output of `signed_square x` should have the same sign as `x`*

- Good :)

- all inputs of a valid implementation $\rightarrow$ returns true

- note that the converse is not true!

Implementing Properties

- *The output of `signed_square x` should have the same sign as `x`*

`fun x → (x > 0) = (signed_square x > 0)`

`fun x → x * signed_square x > 0`

QCheck: PBT for OCaml

- QCheck tests are described by
 - a **generator**: generates random input
 - a **property**: bool-valued function
- OCaml has QCheck
- Python: Hypothesis
- Rust: proptest

Setting Up QCheck

- Install

```
opam install qcheck
```

- Open the QCheck module

```
open QCheck
```

- In utop, before open QCheck

```
#require "qcheck"
```

- In dune file

```
(libraries qcheck)
```

Property Testing

- Making the test

```
open QCheck;  
let test = Test.make ~count:1000 ~name:"signed_square_test"  
  (small_int) (fun x → (x * signed_square x) > 0);;
```

- Running the test

```
QCheck_runner.run_tests ~verbose:true [test];;
```

Implementing Properties Redux

- *The output of `signed_square x` should have the same sign as `x`*
 - Seems like our property implementation was not correct

`fun x → x = 0 || x * signed_square x > 0`

`fun x → (x > 0) = (signed_square x > 0)`

`fun x → compare x 0 = compare (signed_square x) 0`

Let's practice!

- **Some clarifications:**

- Property is **valid** if it describes the intentions of the function (what it's supposed to do)
- Implementation of property is **correct** if it represents the property (regardless of whether the property is good / bad)
- Testing a **correct** representation of a **valid** property can catch bugs, but is not guaranteed to

- **Practice!**

- Fall '24 Quiz 2
- Spring '25 Quiz 2

Problem 4: Property Based Testing

Consider the following function which has a bug in it:

```
1 (* signed_square should take in an int x, square it, but keep the original sign *)  
2 let signed_square x = if x < 0 then (-x) * (-x) else x * x
```

Consider the following property: *the output of signed_square should be greater than or equal to the input*

Is this a valid property? Yes/No: ☐ Y ☒ N

Is the function `fun x -> signed_square x >= x` a correct representation of the property? Yes/No: ☒ Y ☐ N

If we test this property on the provided code, will it ever return false?

Yes: ☐ Y

The property is not valid so the result of testing this property is meaningless: ☒ NA

No: ☐ N

Problem 2: Property Based Testing

Consider the following incorrect `tree_map` function.

```
type tree = Leaf of int | Node of int * tree * tree
```

```
(* has bug(s)! *)
```

```
let rec tree_map f tree = match tree with
```

```
  Leaf(x) -> Leaf(f x)
```

```
  |Node(x,l,r) -> Node(x, tree_map f l, tree_map f r)
```

Consider the following property p about the `tree_map` function:

p : `tree_map` should not change the number of leaves

Using a correct implementation of `tree_map`, this property p should hold true for all valid inputs?

☒ Yes

☐ No

Using our implementation of `tree_map`, this property p would **not** hold true for all valid inputs?

☐ Yes

☒ No

Suppose I encode this property in OCaml to be used in OCaml's QCheck library as the following:

```
let prop f tree = count_leaves tree = count_leaves tree_map tree
```

The above `prop` function is a valid encoding of the property p . ☐ Yes ☒ No