PropR: Property-Based Automatic Program Repair

Matthías Páll Gissurarson*
Chalmers University of Technology
Gothenburg, Sweden
pallm@chalmers.se

Leonhard Applis*
TU Delft
Delft, Netherlands
L.H.Applis@Tudelft.nl

Annibale Panichella
TU Delft
Delft, Netherlands
A.Panichella@Tudelft.nl

Arie van Deursen
TU Delft
Delft, Netherlands
Arie.vanDeursen@Tudelft.nl

David Sands
Chalmers University of Technology
Gothenburg, Sweden
dave@chalmers.se

CHALMERS

TU Delft

ICSE '22, May 21–29, 2022, Pittsburgh, PA, USA

Introduction

Overview - Background

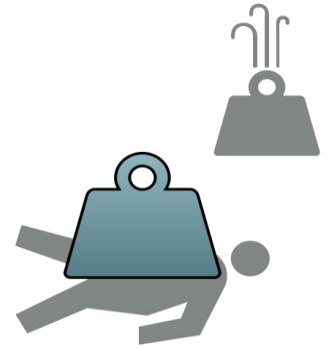
- **Program repair** is a set of techniques used to find patches to repair faulty programs
- **Property-based testing** is a form of randomized testing based on declaring properties
- **Typed-holes** are a way to interact with the compiler by asking about the context of a given location.

```
prop_1 :: Double -> Test
prop_1 x =
    sin x == sin (x+2*π)

prop_2 :: Double -> Test
prop_2 x =
    sin (-1*x ) == -1 * (sin x)

prop_3 :: Test
prop_3 = sin (π/2) == 1

prop_4 :: Test
prop_4 = sin 0 == 0
```

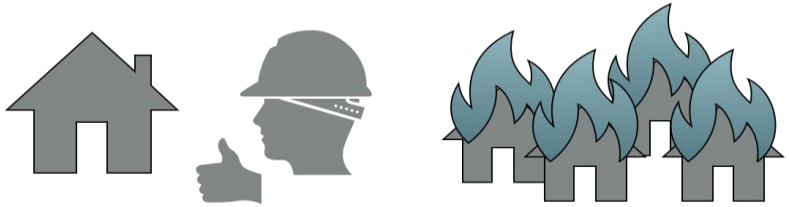
```
GHCi> let degreesToRadians :: Double -> Double
          degreesToRadians d = d * _ / 180

<interactive>:4:30: error:
    • Found hole: _ :: Double
      In the expression: d * _ / 180
    Valid hole fits include
        d :: Double (bound at <interactive>:4:22)
      pi :: forall a. Floating a => a (imported from ‘Prelude’)
```

Challenges in Program Repair

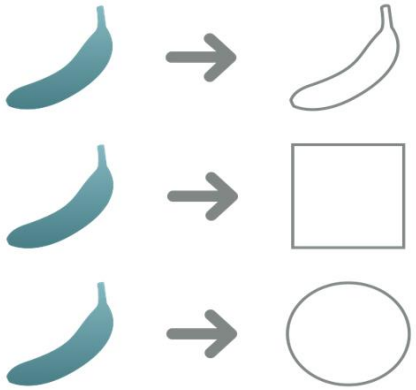
- **Heavy use** of frameworks & meta-programming libraries



- **Overfitting** on "just passing tests"



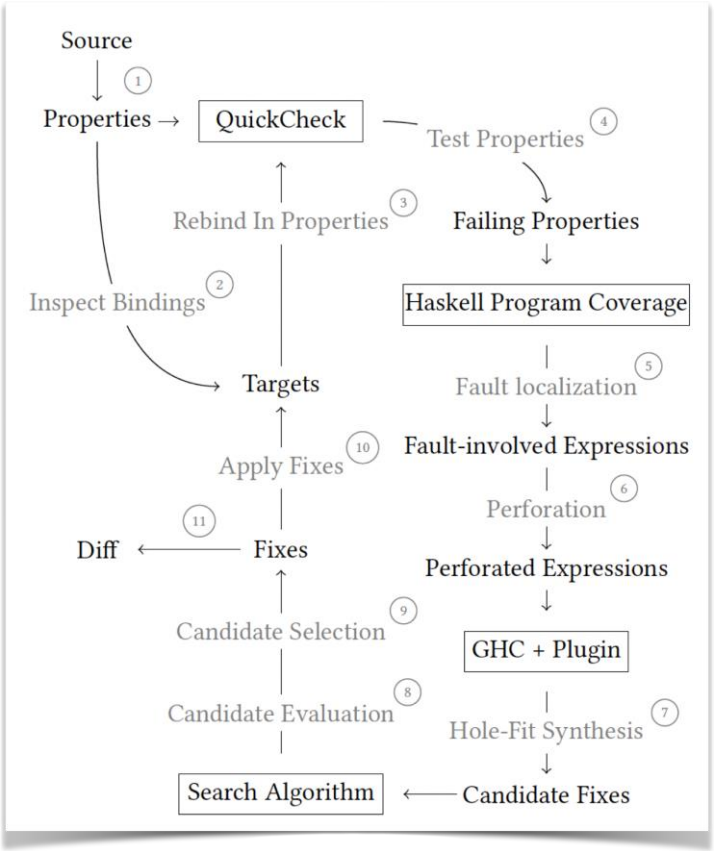
- **Limited search-space**
  (patterns and existing code)



- **Combining** partial solutions

The Propr Way to do it

- **Integrate** with the compiler
- Use **properties** to avoid overfitting
- **Typed-based synthesis** to extend search space
- **Genetic programming** to combine partial solutions



Detailed Approach

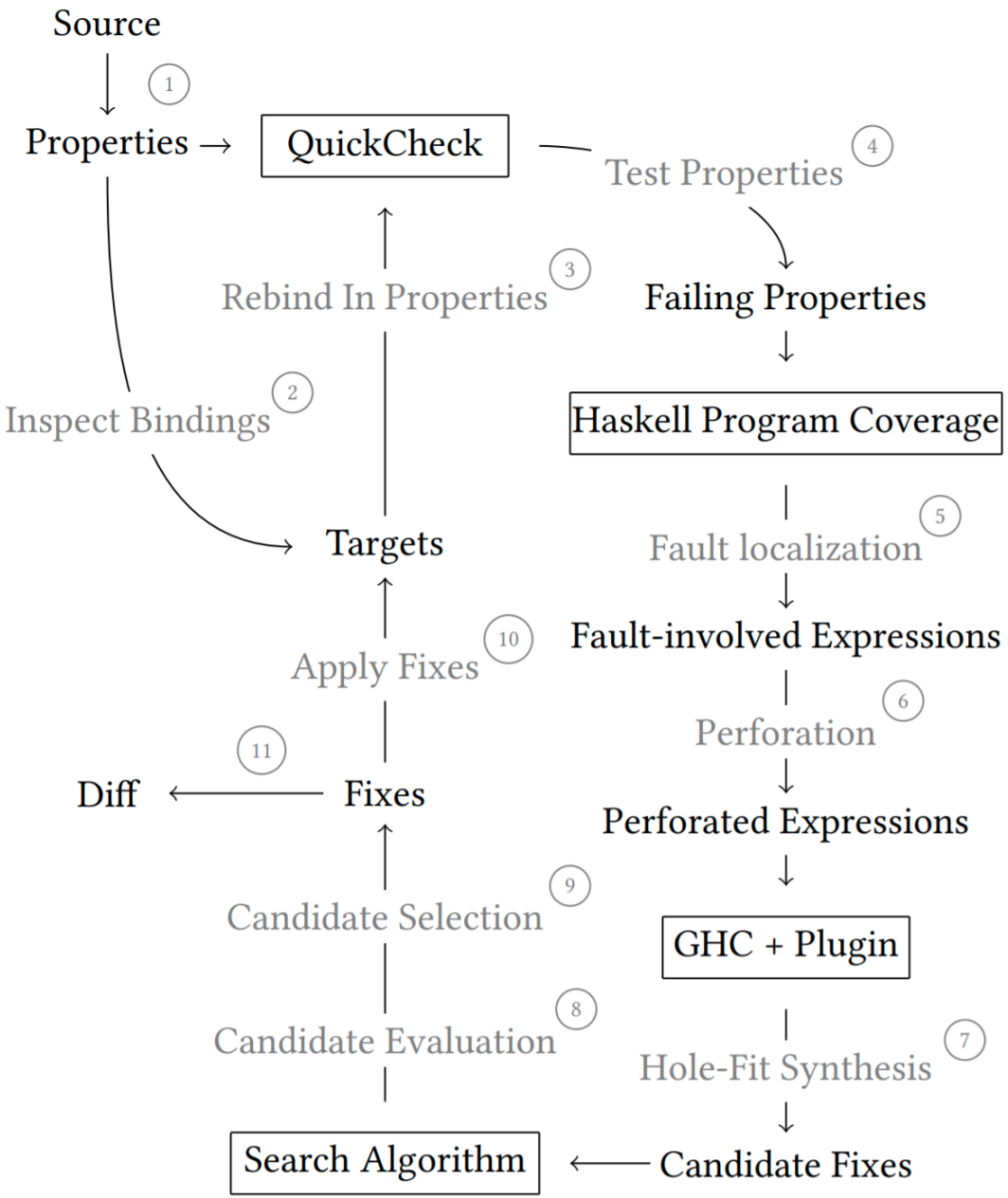


Figure 3: The PROPR test-localize-synthesize-rebind loop

```haskell
gcd' :: Int → Int → Int
gcd' 0 b = gcd' 0 b
gcd' a b | b == 0 = a
gcd' a b = if a > b
           then gcd' (a-b) b
           else gcd' a (b-a)

prop_1 = gcd' 1071 1029 == 21
prop_2 x = gcd' 0 x == x
```

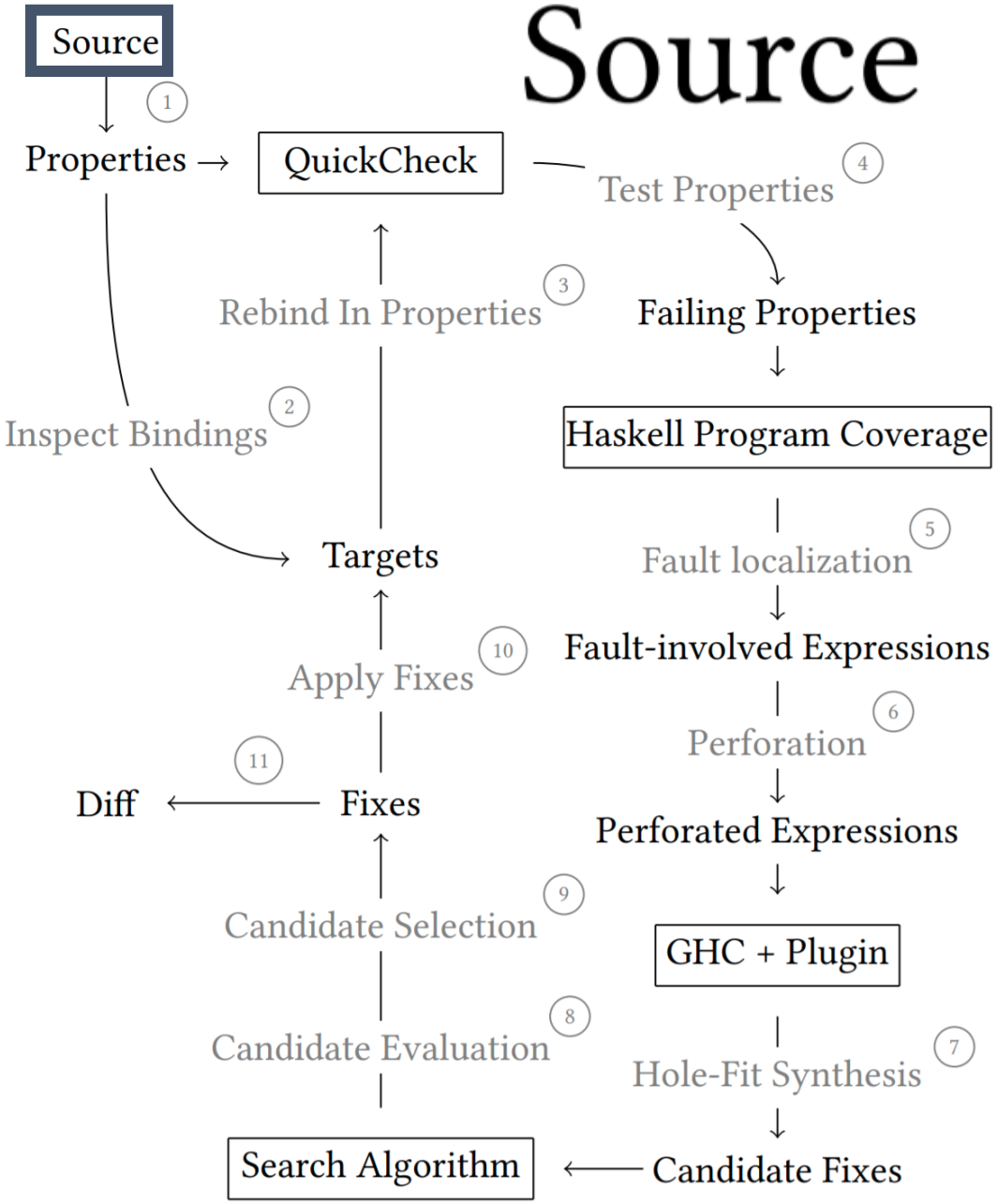


Figure 3: The PropR test-localize-synthesize-rebind loop

```haskell
gcd' :: Int → Int → Int
gcd' 0 b = gcd' 0 b
gcd' a b | b == 0 = a
gcd' a b = if a > b
           then gcd' (a-b) b
           else gcd' a (b-a)

prop_1 = gcd' 1071 1029 == 21
prop_2 x = gcd' 0 x == x
```

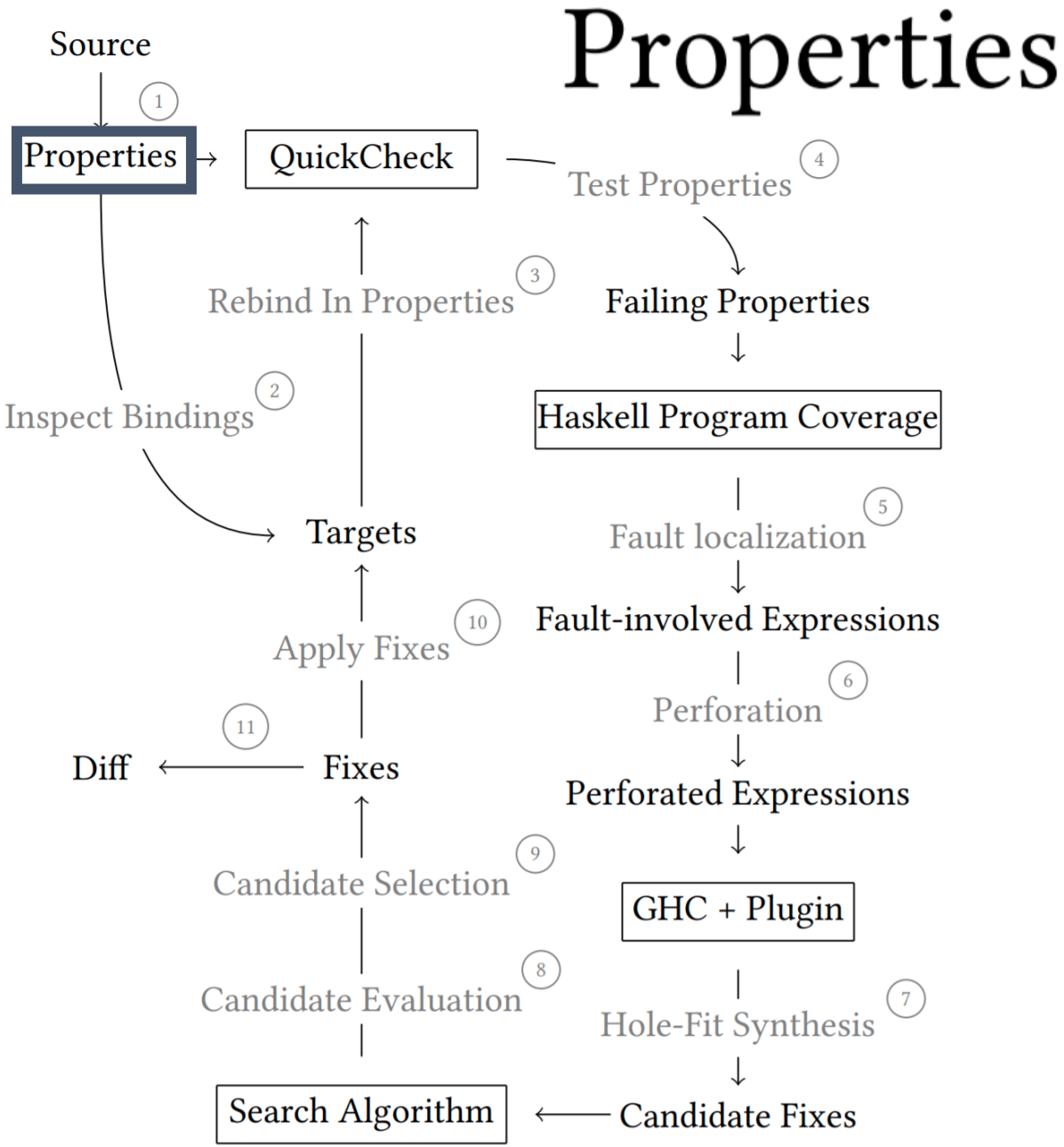


Figure 3: The PROPR test-localize-synthesize-rebind loop

```haskell
gcd' :: Int → Int → Int
gcd' 0 b = gcd' 0 b
gcd' a b | b == 0 = a
gcd' a b = if a > b
           then gcd' (a-b) b
           else gcd' a (b-a)
```

```haskell
prop_1 = gcd' 1071 1029 == 21
prop_2 x = gcd' 0 x == x
```

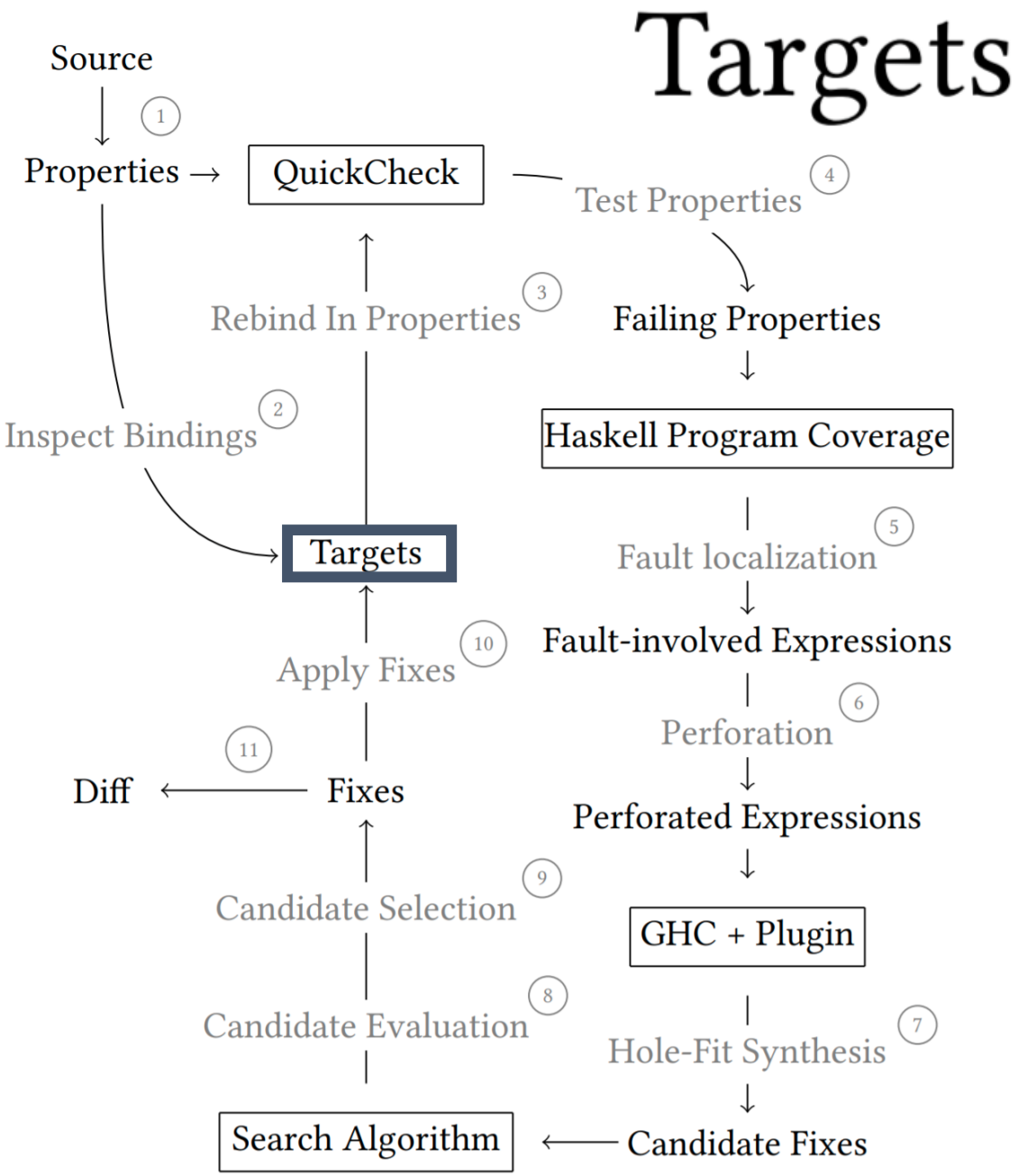
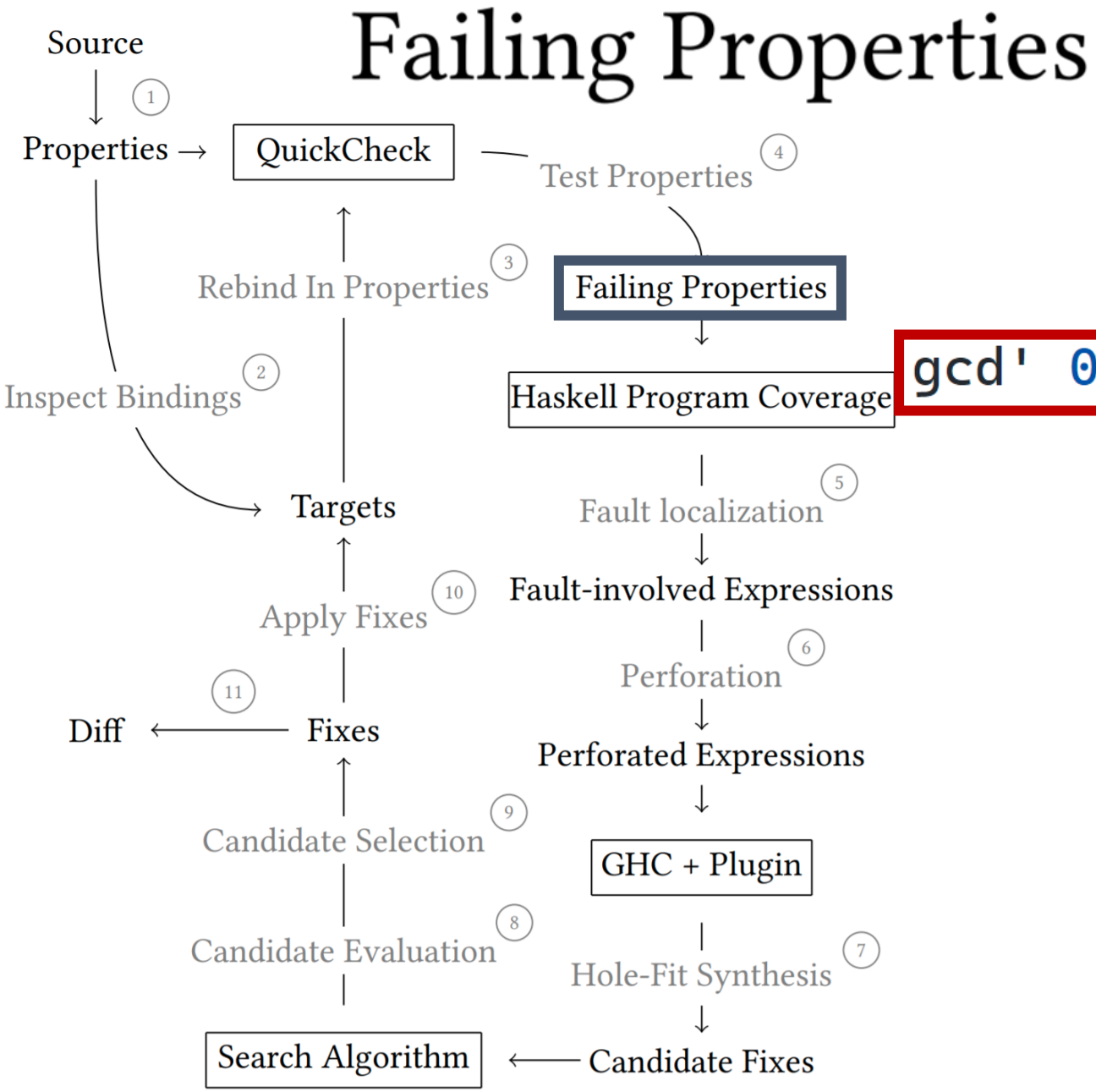


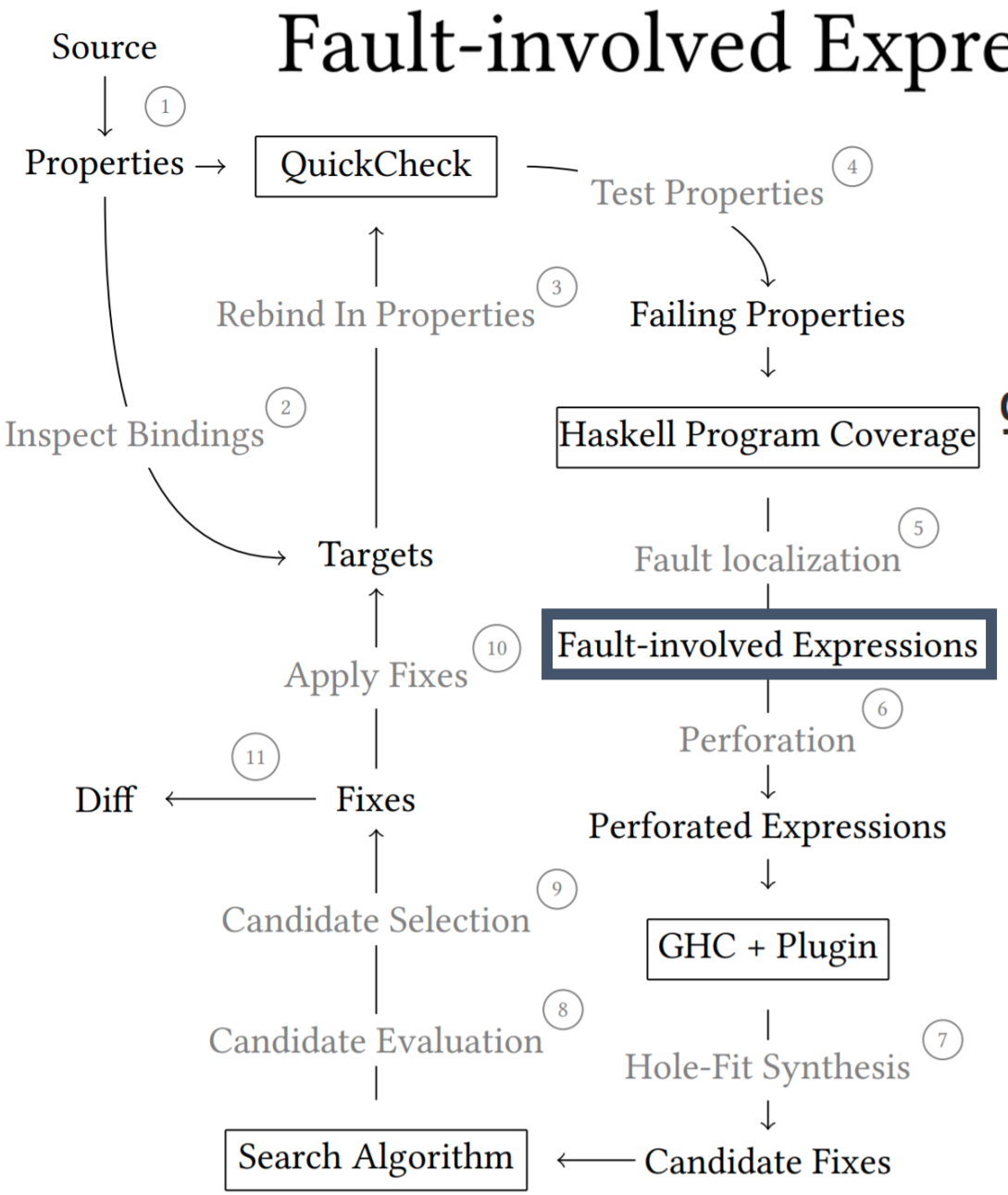
Figure 3: The PROPR test-localize-synthesize-rebind loop

```haskell
gcd' :: Int → Int → Int
gcd' 0 b = gcd' 0 b
gcd' a b | b == 0 = a
gcd' a b = if a > b
           then gcd' (a-b) b
           else gcd' a (b-a)

prop_1 = gcd' 1071 1029 == 21
prop_2 x = gcd' 0 x == x
```




```haskell
gcd' :: Int → Int → Int
gcd' 0 b = gcd' 0 b
gcd' a b | b == 0 = a
gcd' a b = if a > b
           then gcd' (a-b) b
           else gcd' a (b-a)

prop_1 = gcd' 1071 1029 == 21
prop_2 x = gcd' 0 x == x
```

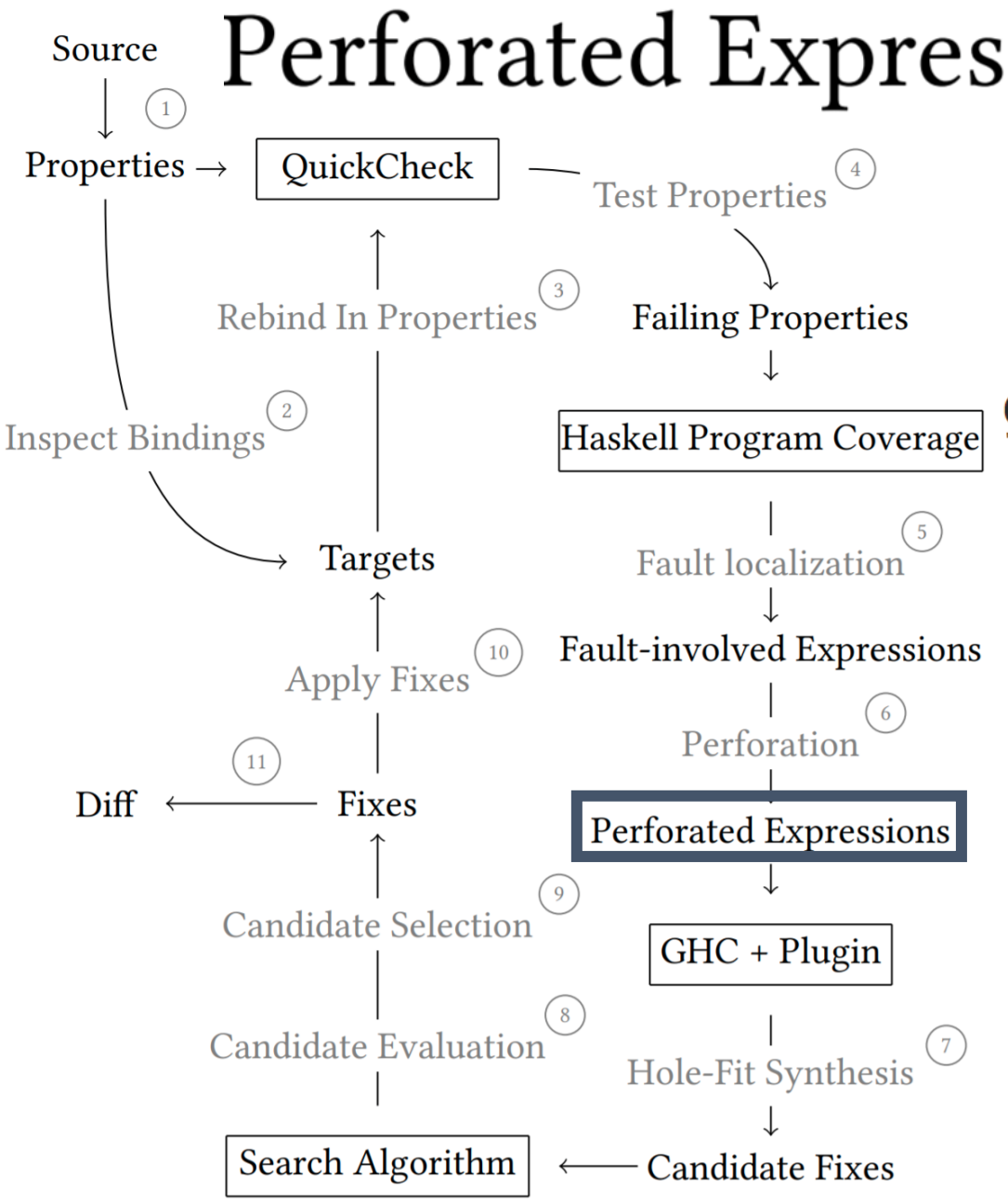
Figure 3: The PropR test-localize-synthesize-rebind loop




```
gcd' 0 0
```

```
gcd' :: Int → Int → Int
gcd' 0 b = gcd' 0 b
gcd' a b | b == 0 = a
gcd' a b = if a > b
           then gcd' (a-b) b
           else gcd' a (b-a)

prop_1 = gcd' 1071 1029 == 21
prop_2 x = gcd' 0 x == x
```

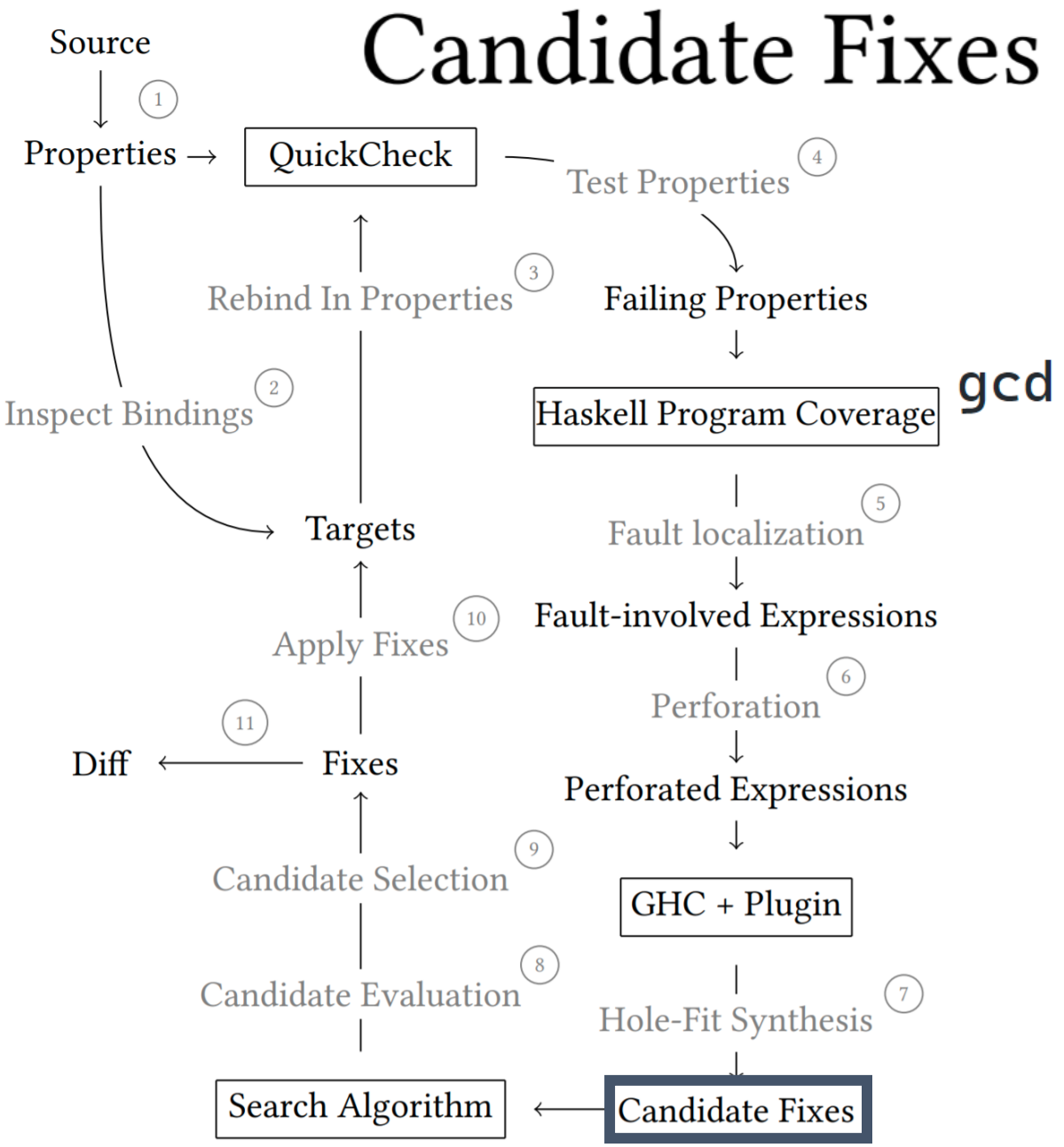
Figure 3: The PropR test-localize-synthesize-rebind loop




```
gcd' 0 0
```

```
gcd' :: Int → Int → Int
gcd' 0 b = _
gcd' a b | b == 0 = a
gcd' a b = if a > b
           then gcd' (a-b) b
           else gcd' a (b-a)
prop_1 = gcd' 1071 1029 == 21
prop_2 x = gcd' 0 x == x
```

Figure 3: The PROPR test-localize-synthesize-rebind loop

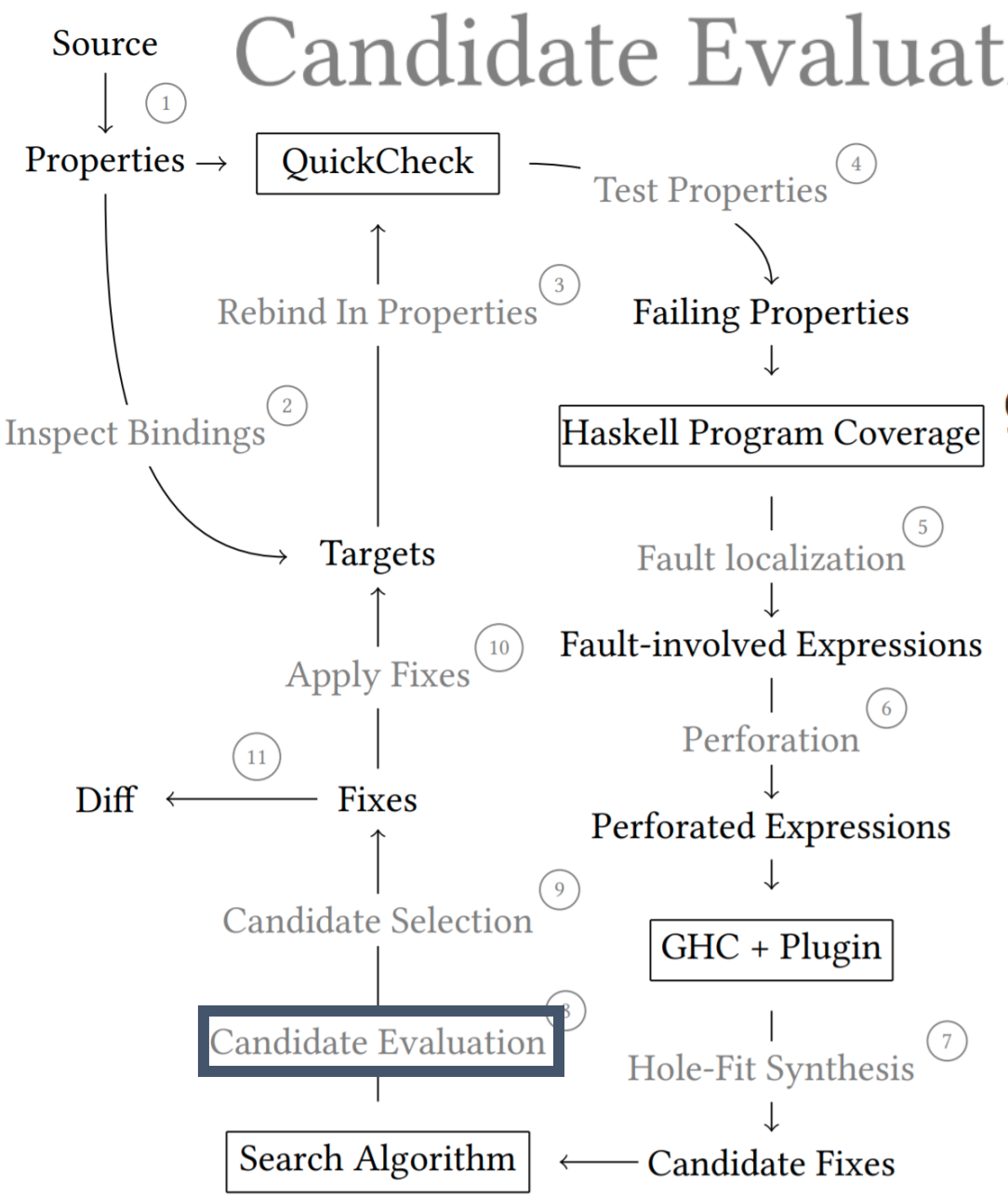



```
gcd' 0 0
```

```
gcd' :: Int → Int → Int
gcd' 0 b = _
gcd' a b | b == 0 = a
gcd' a b = if a > b
           then gcd' (a-b) b
           else gcd' a (b-a)
prop_1 = gcd' 1071 1029 == 21
prop_2 x = gcd' 0 x == x
```

```
minBound :: forall a. Bounded a ⇒ a
0, 1071, 1029, 21 :: Int
b :: Int
```

Figure 3: The PROPR test-localize-synthesize-rebind loop




```
gcd' 0 0
```

```
gcd' :: Int → Int → Int
gcd' 0 b = b
gcd' a b | b == 0 = a
gcd' a b = if a > b
           then gcd' (a-b) b
           else gcd' a (b-a)
prop_1 = gcd' 1071 1029 == 21
prop_2 x = gcd' 0 x == x
```

```
minBound :: forall a. Bounded a ⇒ a
0, 1071, 1029, 21 :: Int
b :: Int
```

Figure 3: The PropR test-localize-synthesize-rebind loop

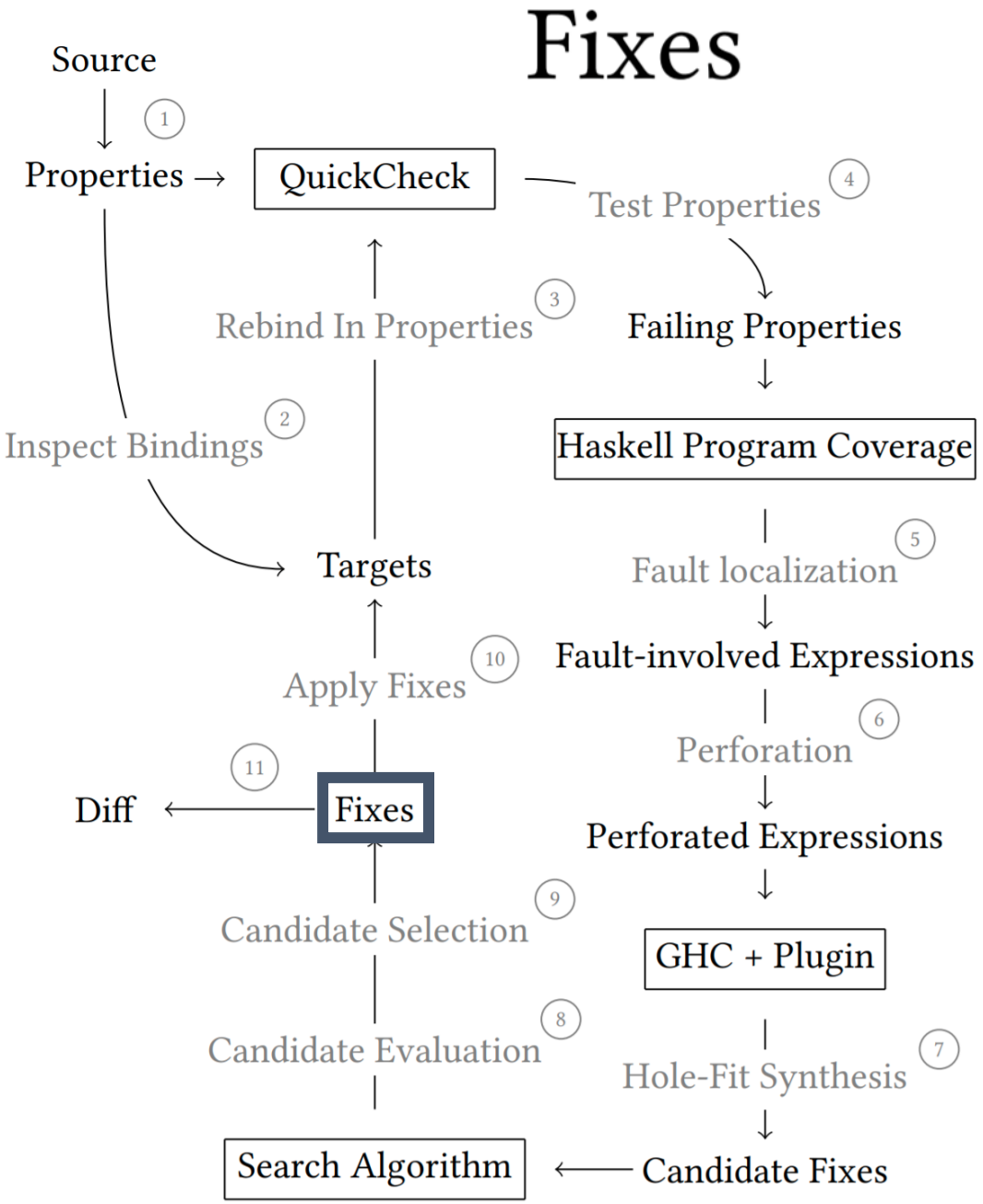


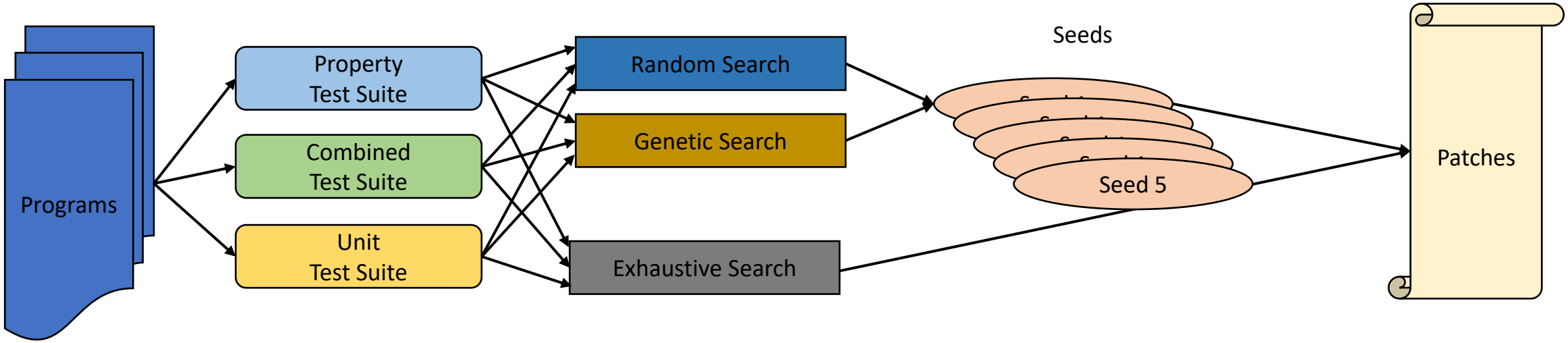
Figure 3: The PROPR test-localize-synthesize-rebind loop

```
--- a/examples/BrokenGCD.hs
+++ b/examples/BrokenGCD.hs
@@ -16,3 +16,3 @@ gcd' 0 b = gcd' 0 b
-gcd' 0 b = gcd' 0 b
+gcd' 0 b = b
 gcd' a b | b == 0 = a
```

Demo

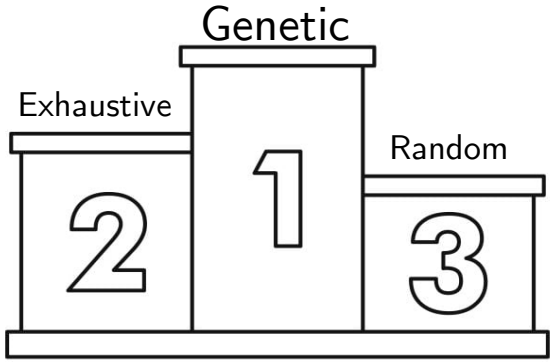
Experiment & Results

Experiment Setup



- 30 **student programs** from Chalmers
- Each fails 1 or more of 23 properties and 20 unit tests
- 3 test-suites, 3 search algorithms, 5 Seeds amount to 31 Configurations
- 5 minute search budget per configuration

Results



- **Genetic search** found all patches of exhaustive search
- **Random search** did not find any patches (search space too big?)
- The **combined** test-suite found patches the fastest

- 13 / 30 programs **patched**, 213 patches in total
- 63-85% **overfit** (manual analysis)
- 4 programs **effectively** repaired (13% repair rate)

Manual Analysis

```
diff --git a//input/expr_both.hs b//input/expr_both.hs
--- a//input/expr_both.hs
+++ b//input/expr_both.hs
@@ -44,6 +44,6 @@ showExpr Var = "x"
 showExpr Var = "x"
 showExpr (Num i) = show i
-showExpr (NormOp Add x y) = showExpr x ++ "+" ++ showExpr y
+showExpr (NormOp Add x y) = ((filter (not . isSpace)) (showExpr x ++ "+" ++ showExpr y))
 showExpr (NormOp Mul x y) = showFactor x ++ "*" ++ showFactor y
 showExpr (TrigExpr Sin expr) = "sin " ++ "(" ++ showExpr expr ++ ")"
 showExpr (TrigExpr Cos expr) = "cos " ++ "(" ++ showExpr expr ++ ")"
```

A seemingly good patch?

- Overfitting **clarifies** test-suite issues
- Test-suite issues **persist** in the combined test-suite
- **Sophisticated** patches
- Overfit patches give **hint** at what needs to be fixed

Future Work

- Industry-based **datasets** and **configurations**
- Improved **fault-localization** and **synthesis**
- **Automated feedback** systems for GitHub and teaching
- **Co-evolution** of tests and code

See you at ICSE!

Questions? Ask us
@tritlo and
@lapplislazuli on Twitter

PROPR is available at:

`github.com/Tritlo/PropR`