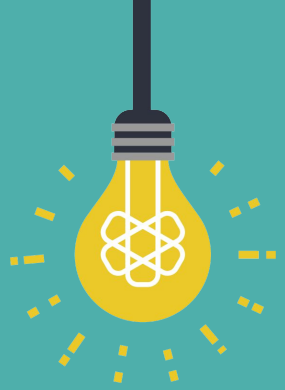


Generating Unit Tests

with Automated Source Code Analysis

symflower

AUTOMATING QUALITY ASSURANCE

Agenda

1. Background Information (5 min)
2. Demonstration (10 min)

Symflower Company

symflower

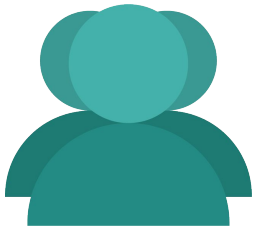
- Startup based in Linz, Austria, EU
- Vision: Complete automation of software QA

Founders

Evelyn Haslinger

Markus Zimmermann

Team



Currently a team of 13

Technology and Product

symflower

Symflower completely **automatically finds**, writes, runs and analyses **all relevant unit tests** revealing bugs, security issues and performance problems.

→ Reduce development and maintenance time

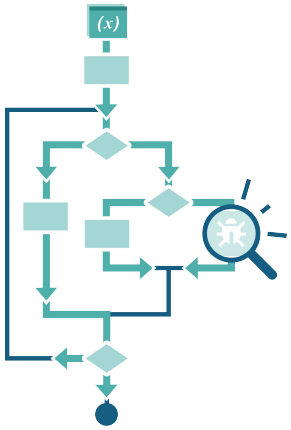
→ Increase quality of your software and tests

Technology and Product

symflower

Our technology to generate test candidates:

- Symbolic execution (SE)
  - Checks **every** functionality
  - Computes **targeted** test cases
  - Reaches **highest** test coverage
  - **Finds bugs automatically**

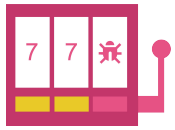


Alternatives:

Boundary value analysis

- Gives **very low coverage**

Fuzzing



- Coverage depends on **luck**

Symbolic Execution

symflower

Executing a program symbolically means, that rather than operating on concrete values, one is **operating on symbolic values** considering **all possible execution paths <u>at once</u>**. Constraint solvers are used to get concrete values fulfilling the constraints that describe a path.

```
func f(a int, b int) int {
  x, y := 1, 0
  if a != 0 {
    y = x + 3
    if b == 0
        x = 2 * (a + b)
  }
  return (a + b) / (x - y)
}
```

Assignments

Path split

Path constraints

Request to Constraint solver

Solution from constraint solver

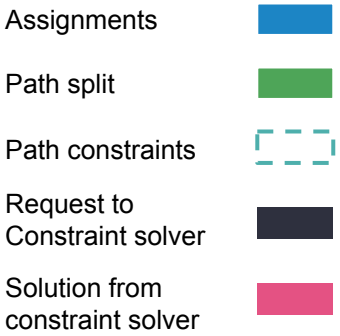
Symbolic Execution

symflower

Executing a program symbolically means, that rather than operating on concrete values, one is **operating on symbolic values** considering **all possible execution paths <u>at once</u>**. Constraint solvers are used to get concrete values fulfilling the constraints that describe a path.

```
func f(a int, b int) int {
  x, y := 1, 0
  if a != 0 {
    y = x + 3
    if b == 0
        x = 2 * (a + b)
  }
  return (a + b) / (x - y)
}
```

Symbolic execution to find divisions by zero:



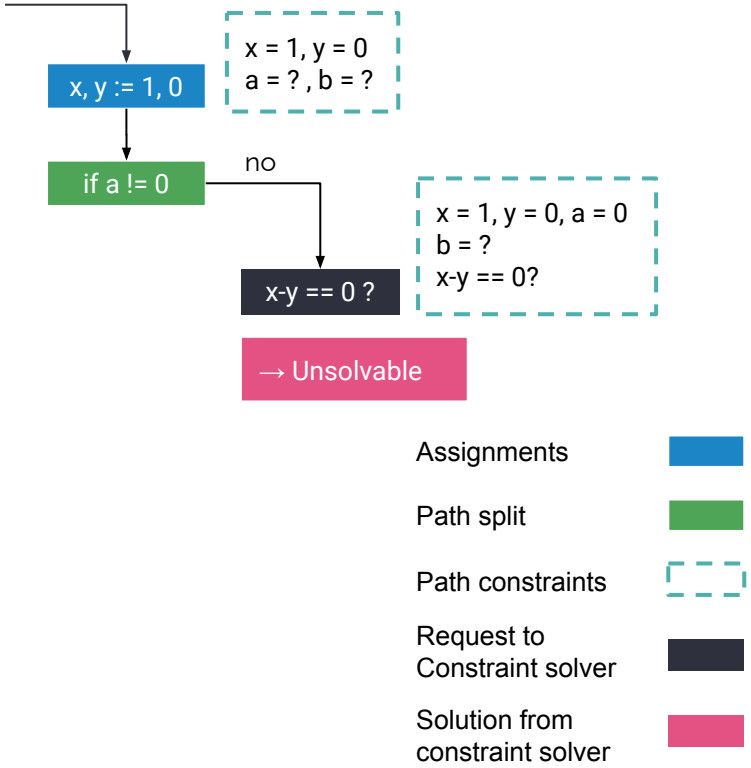
Symbolic Execution

symflower

Executing a program symbolically means, that rather than operating on concrete values, one is **operating on symbolic values** considering **all possible execution paths <u>at once</u>**. Constraint solvers are used to get concrete values fulfilling the constraints that describe a path.

```go
func f(a int, b int) int {
  x, y := 1, 0
  if a != 0 {
    y = x + 3
    if b == 0
        x = 2 * (a + b)
  }
  return (a + b) / (x - y)
}
```

Symbolic execution to find divisions by zero:



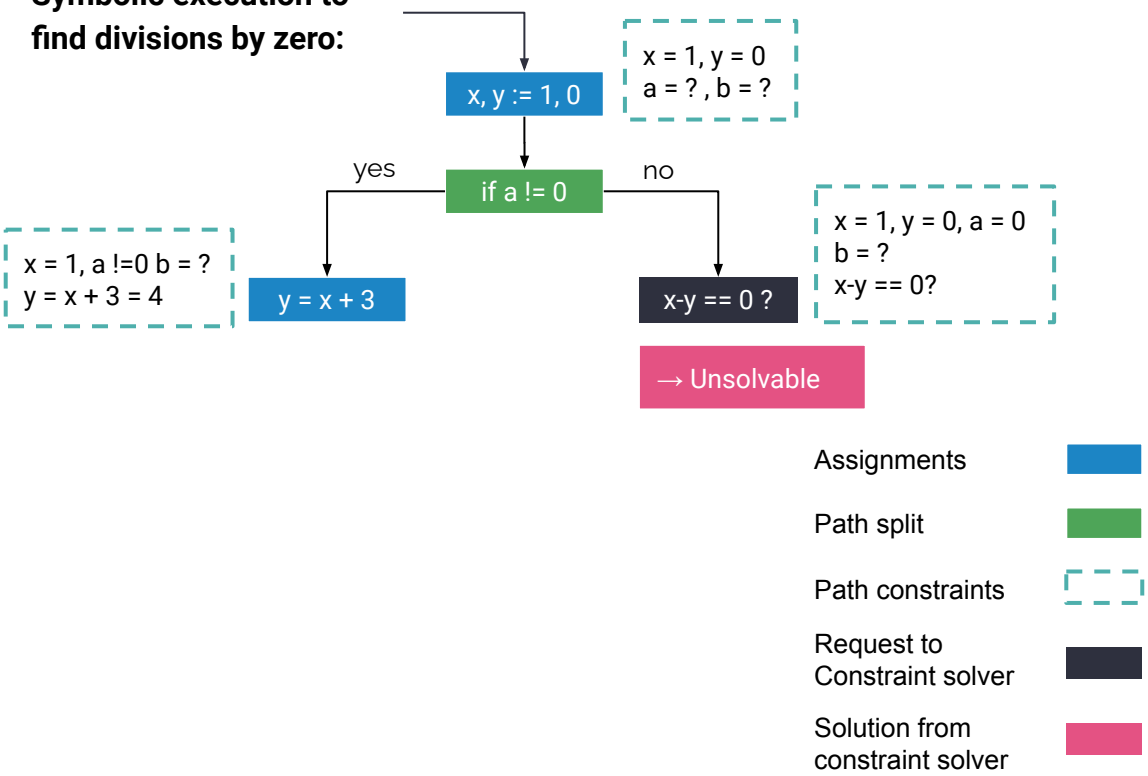
Symbolic Execution

symflower

Executing a program symbolically means, that rather than operating on concrete values, one is **operating on symbolic values** considering **all possible execution paths <u>at once</u>**. Constraint solvers are used to get concrete values fulfilling the constraints that describe a path.

```
func f(a int, b int) int {
  x, y := 1, 0
  if a != 0 {
    y = x + 3
    if b == 0
        x = 2 * (a + b)
  }
  return (a + b) / (x - y)
}
```

Symbolic execution to find divisions by zero:



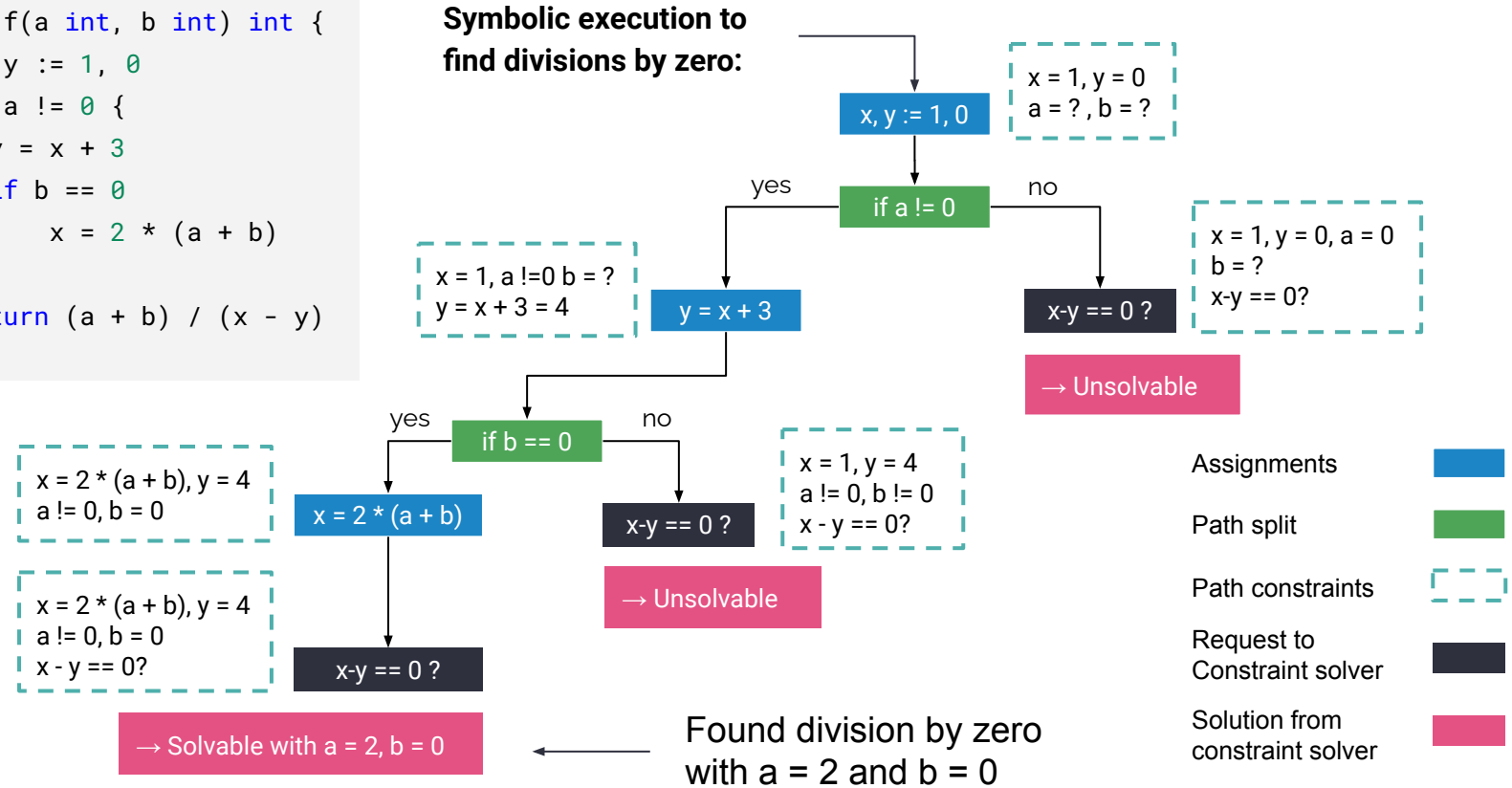
Symbolic Execution

symflower

Executing a program symbolically means, that rather than operating on concrete values, one is **operating on symbolic values** considering **all possible execution paths <u>at once</u>**. Constraint solvers are used to get concrete values fulfilling the constraints that describe a path.

```go
func f(a int, b int) int {
  x, y := 1, 0
  if a != 0 {
    y = x + 3
    if b == 0
        x = 2 * (a + b)
  }
  return (a + b) / (x - y)
}
```

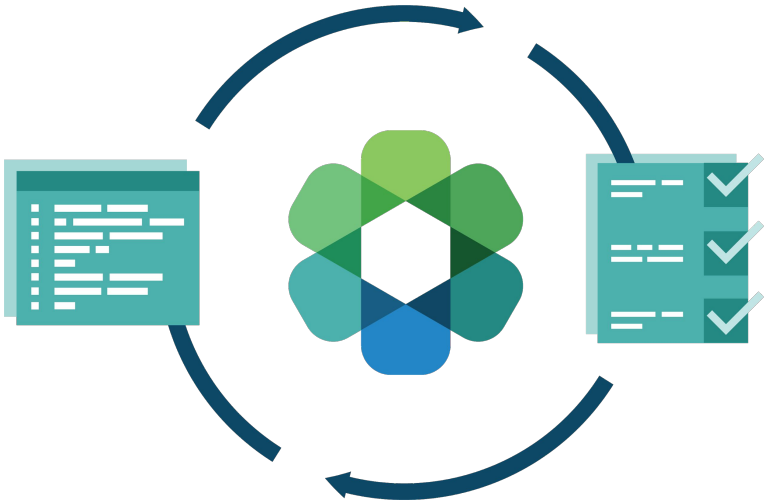
Symbolic execution to find divisions by zero:



Symflower the Product

symflower

➔ Let’s take a look



symflower

AUTOMATING QUALITY ASSURANCE

Evelyn Haslinger eh@symflower.com

Markus Zimmermann mz@symflower.com