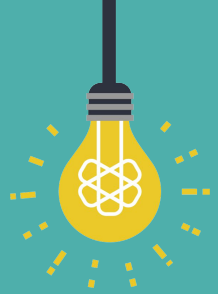


Debugging With Symflower

Using Unit Tests to Find and Fix Bugs

symflower

AUTOMATING QUALITY ASSURANCE

debugging.symflower.com

“Program testing can be used to show the presence of bugs, but never to show their absence.”

Dijkstra

Dijkstra might be right, but...

Agenda

1. What is Symflower?
2. Workshop
3. Discussion and Questions

1 What is Symflower?

History Excursion

Product to automatically generate unit tests to validate and reveal bugs and security issues

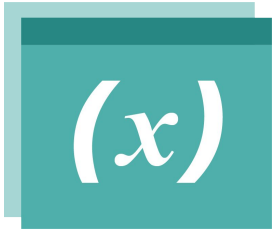
symflower

AUTOMATING QUALITY ASSURANCE

Symbolic Execution

symflower

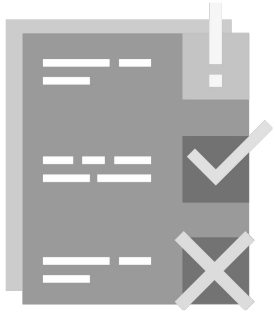
Target Program



Symbolic Exploration

Reasoning Engine

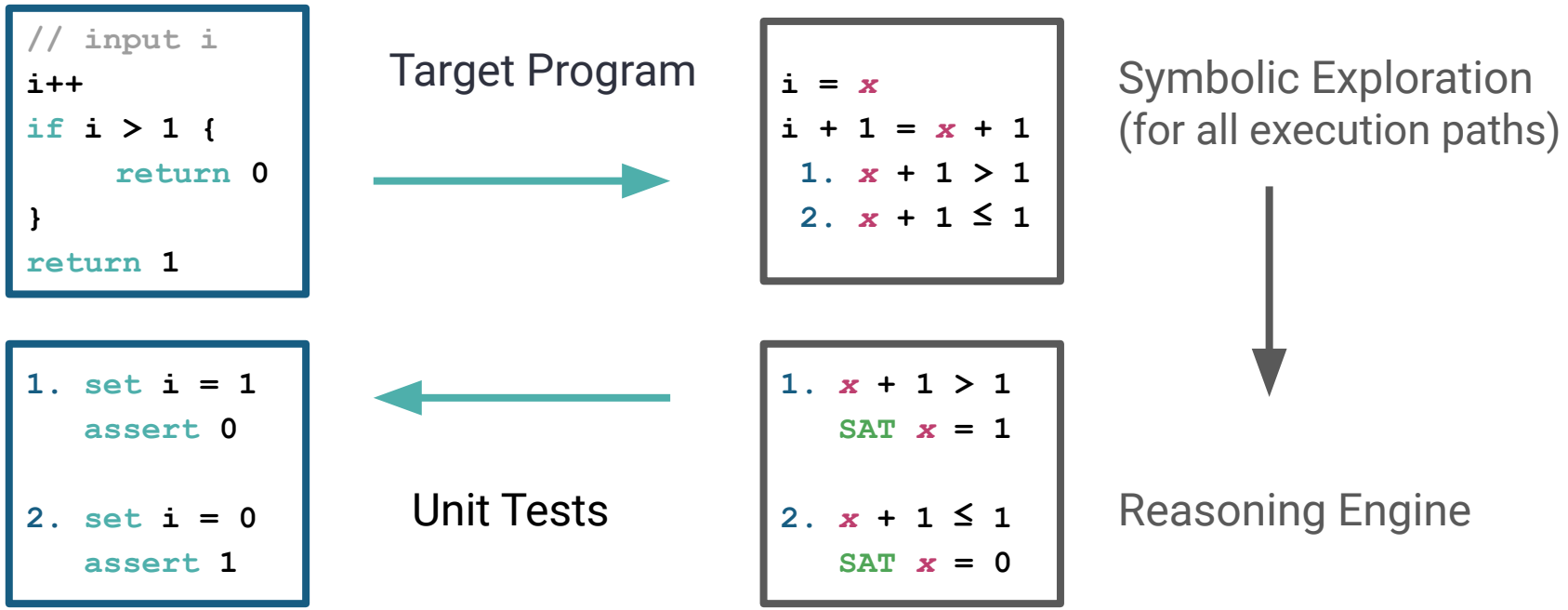
Unit Tests



"Symbolic Execution and Program Testing", James C. King (1976)

Symbolic Execution

symflower



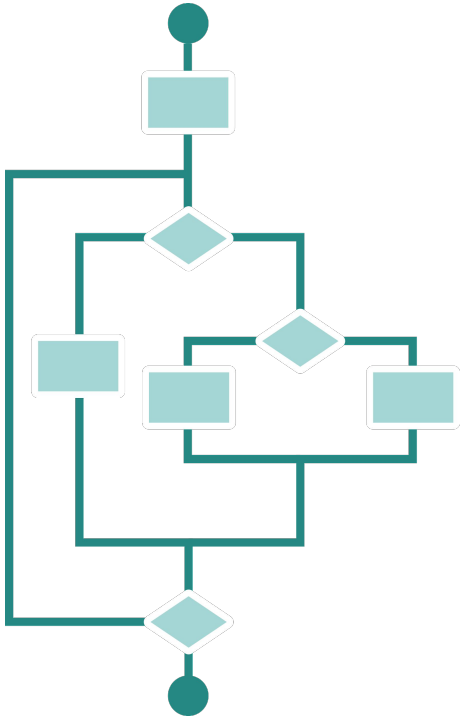
"Symbolic Execution and Program Testing", James C. King (1976)

More Details

symflower

- Overview
- Symbolic Execution
- Other Methods

Original Paper by James C. King



Institute for
Formal Models and Verification

symflower

Former Students / Founders

- DI Evelyn Haslinger, BSc
- DI Markus Zimmermann, BSc
- Simon Bauer, BSc

Former Lecturers

- Univ.-Prof.^{in} Dr.^{in} Martina Seidl
- Univ.-Prof. Dr. Armin Biere

The Company

symflower

Unit test case generation, problem detection, assisted debugging, high code coverage, real-time feedback…

- founded in 2018
- 10x employees
- several awards

symflower.com

2 Workshop

Debugging with Unit Tests

Follow along yourself!

go to **get.symflower.com**

for Linux, macOS and Windows

Hashing

symflower

$$h : X \longrightarrow Y$$

Security, Cryptography, Compression, Efficient Algorithms
Checksums, Error Correction, Fast Databases

Our Hash Function

symflower

```
func babyHash(in int) int {
    div := 3
    for i := 0; i < 3; i++ {
        in = in / div
        div = div + in
    }
    return div
}
```

Source Code: debugging.symflower.com

Examples

symflower

The Story

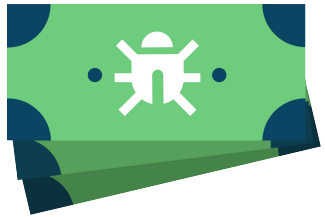
symflower

⟶ GitHub or r/cybersecurity

Bug Report

symflower

```
func babyHash(in int) int {
    div := 3
    for i := 0; i < 3; i++ {
        in = in / div
        div = div + in
    }
    return div
}
```



-17

Crashes

Debugging Plan

symflower

1. Isolate Problem
2. Simple **Reproducer**
3. **Debug** Problem
4. **Fix** (Watch for **Regressions**)

Simplified Version of **TRAFFIC** Principle

Generate Unit Tests

symflower

`symflower`

- freeze behavior
- edge cases

```
func TestSymflowerBabyHash1(t *testing.T) {
    in := -9
    babyHash(in)
}

func TestSymflowerBabyHash2(t *testing.T) {
    in := 0
    actual := babyHash(in)
    expected := 3
    assert.Equal(t, expected, actual)
}
```

Debugging – 9

symflower

```
func babyHash(in int) int {
    div := 3
    for i := 0; i < 3; i++ {
        in = in / div
        div = div + in
    }
    return div
}
```

$in = -9$

$in_0 = -9/3 = -3$

$div_0 = 3-3 = 0$

Division by Zero

Debugging -17

symflower

```
func babyHash(in int) int {
    div := 3
    for i := 0; i < 3; i++ {
        in = in / div
        div = div + in
    }
    return div
}
```

$in = -17$

$in_0 = -17/3 = -5$

$div_0 = 3-5 = -2$

$in_1 = -5/-2 = 2$

$div_1 = -2+2 = 0$

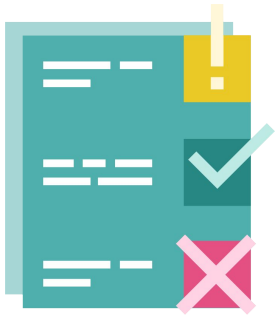
Potential Fix

symflower

```
func babyHash(in int) int {
    div := 3
    for i := 0; i < 3; i++ {
        in = div / in
        div = div + in
    }
    return div
}
```

div = 0 ✓

in = 0 ✗



Tests catch Regression

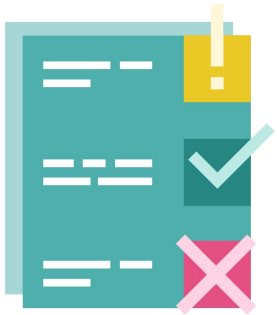
symflower

```
func TestSymflowerBabyHash1(t *testing.T) {
    in := -9
    babyHash(in)
}

func TestSymflowerBabyHash2(t *testing.T) {
    in := 0
    actual := babyHash(in)
    expected := 3
    assert.Equal(t, expected, actual)
}
```

div = 0 ✓

in = 0 ✗



Correct Fix

symflower

```
func babyHash(in int) int {
    div := 3
    for i := 0; i < 3; i++ {
        if (div == 0) {
            div = 3
        }
        in = in / div
        div = div + in
    }
    return div
}
```

div = 0 ✓

in = 0 ✓

Generate Unit Tests

symflower

symflower

- `div = 0` ✓
- one iteration

```
func TestSymflowerBabyHash1(t *testing.T) {
    in := -15
    actual := babyHash(in)
    expected := 3
    assert.Equal(t, expected, actual)
}
func TestSymflowerBabyHash2(t *testing.T) {
    in := 0
    actual := babyHash(in)
    expected := 3
    assert.Equal(t, expected, actual)
}
```


WHEN YOU FIXED ONE BUG
AND THE BUG TRACKER
STILL HAS TEN MORE
imgflip.com

Recap

symflower

No one writes bug-free code!

⇨ Workflows and tools to **enhance debugging**

Unit Tests give **repeatable, simple checks** and **prevent regressions** (+ catch bugs before production)

3 Discussion

Questions?

symflower
AUTOMATING QUALITY ASSURANCE

symflower

AUTOMATING QUALITY ASSURANCE

Simon Bauer | Johannes Altmanninger

office@symflower.com

debugging.symflower.com