RTL-Repair: Fast Symbolic Repair of Hardware Design Code

Kevin Laeufer
laeufer@eecs.berkeley.edu
University of California, Berkeley
Berkeley, CA, USA

Brandon Fajardo*
brfajardo@berkeley.edu
University of California, Berkeley
Berkeley, CA, USA

Abhik Ahuja*
ahujaabhik@berkeley.edu
University of California, Berkeley
Berkeley, CA, USA

Vighnesh Iyer
vighnesh.iyer@eecs.berkeley.edu
University of California, Berkeley
Berkeley, CA, USA

Borivoje Nikolić
bora@eecs.berkeley.edu
University of California, Berkeley
Berkeley, CA, USA

Koushik Sen
ksen@eecs.berkeley.edu
University of California, Berkeley
Berkeley, CA, USA

Paper PDF

Code on Github

Artifacts Available acm V1.1

Artifacts Evaluated acm Functional V1.1

Results Reproduced acm V1.1

Automated RTL Repair

RTL Design

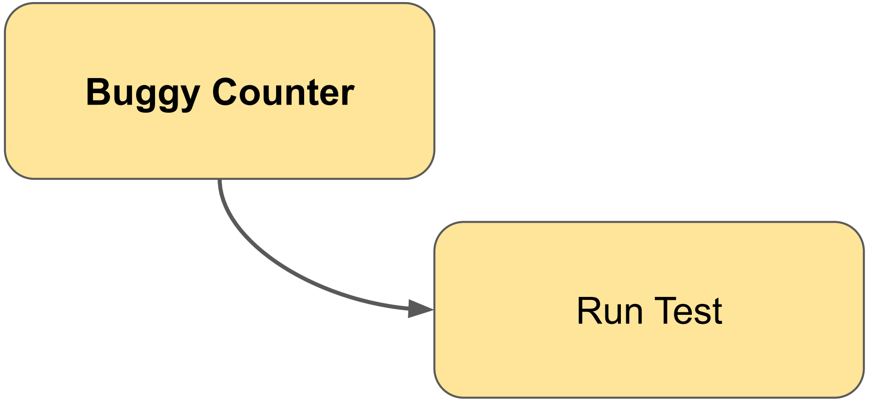
```
class Counter extends Module {
 val io = IO(new CounterIO)

 val count = RegInit(0.U(4.W))
 val overflow = RegInit(false.B)
 when(io.enable) {
   count := count + 1.U
 }
 when(count === "b1111".U) {
   overflow := true.B
 }

 io.count := count
 io.overflow := overflow
}
```

Automated RTL Repair

Buggy Counter

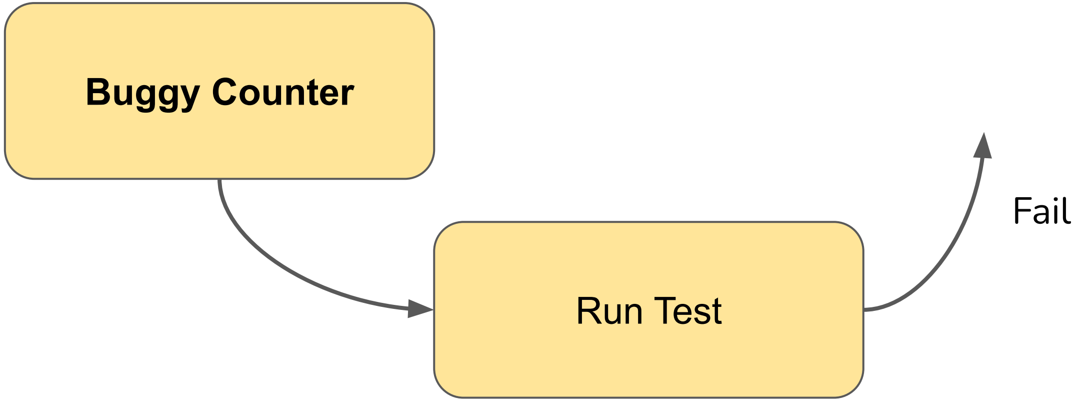
```
class Counter extends Module {
 val io = IO(new CounterIO)

 val count = RegInit(0.U(4.W))
 val overflow = RegInit(false.B)
 when(io.enable) {
   count := count + 1.U
 }
 when(count === "b0111".U) {
   overflow := true.B
 }

 io.count := count
 io.overflow := overflow
}
```

Automated RTL Repair



```
class Counter extends Module {
 val io = IO(new CounterIO)

 val count = RegInit(0.U(4.W))
 val overflow = RegInit(false.B)
 when(io.enable) {
   count := count + 1.U
 }
 when(count === "b0111".U) {
   overflow := true.B
 }

 io.count := count
 io.overflow := overflow
}
```

Automated RTL Repair



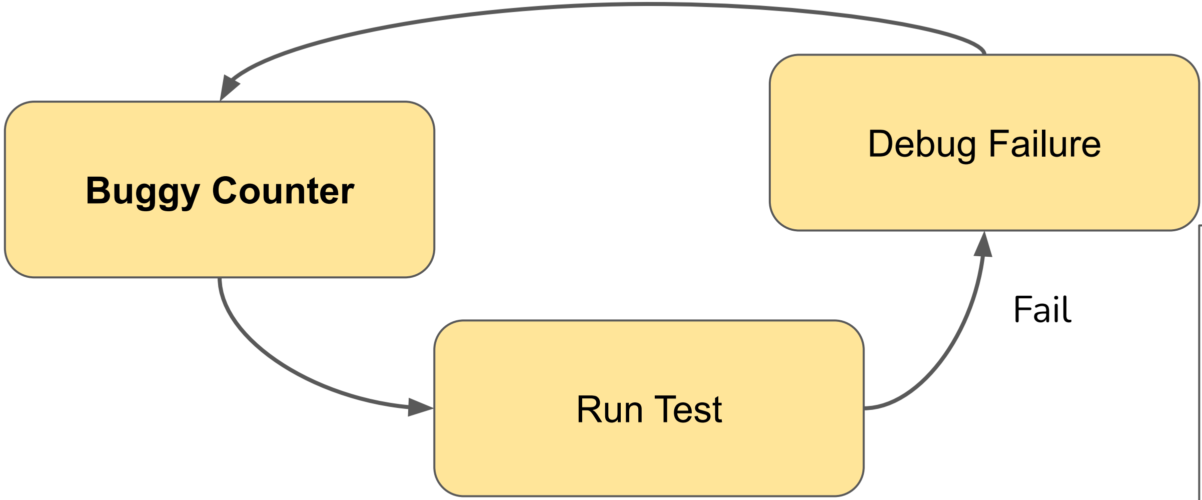
Buggy Counter

Run Test

Fail

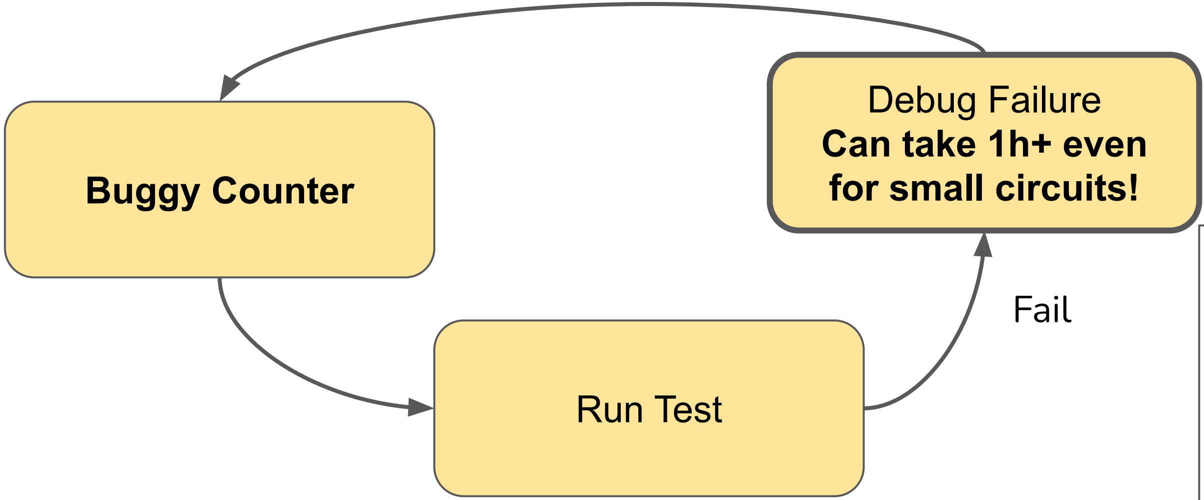
```
class Counter extends Module {
 val io = IO(new CounterIO)

 val count = RegInit(0.U(4.W))
 val overflow = RegInit(false.B)
 when(io.enable) {
   count := count + 1.U
 }
 when(count === "b0111".U) {
   overflow := true.B
 }

 io.count := count
 io.overflow := overflow
}
```

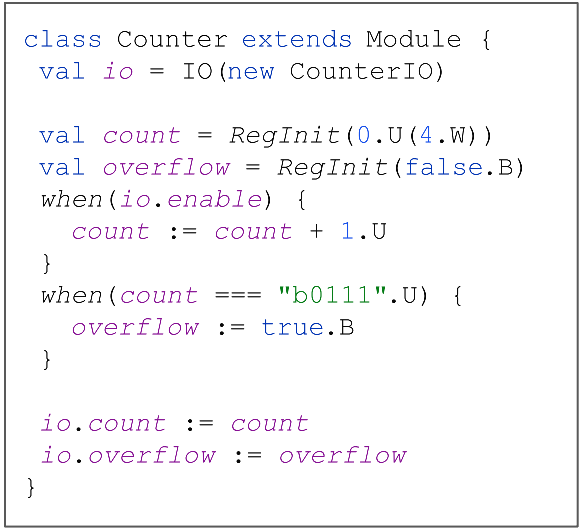
Automated RTL Repair



```
class Counter extends Module {
 val io = IO(new CounterIO)

 val count = RegInit(0.U(4.W))
 val overflow = RegInit(false.B)
 when(io.enable) {
   count := count + 1.U
 }
 when(count === "b0111".U) {
   overflow := true.B
 }

 io.count := count
 io.overflow := overflow
}
```

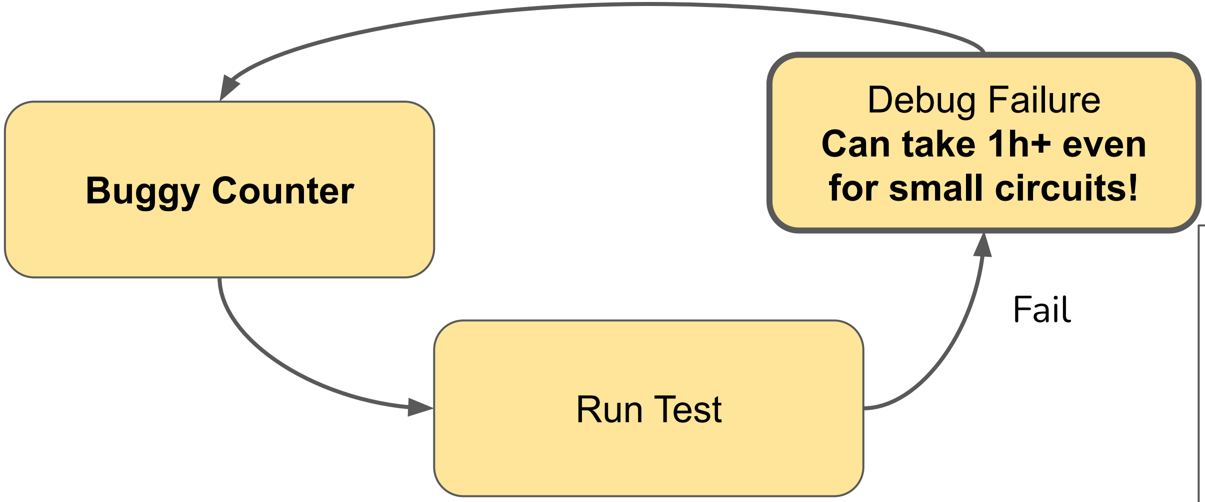
Automated RTL Repair





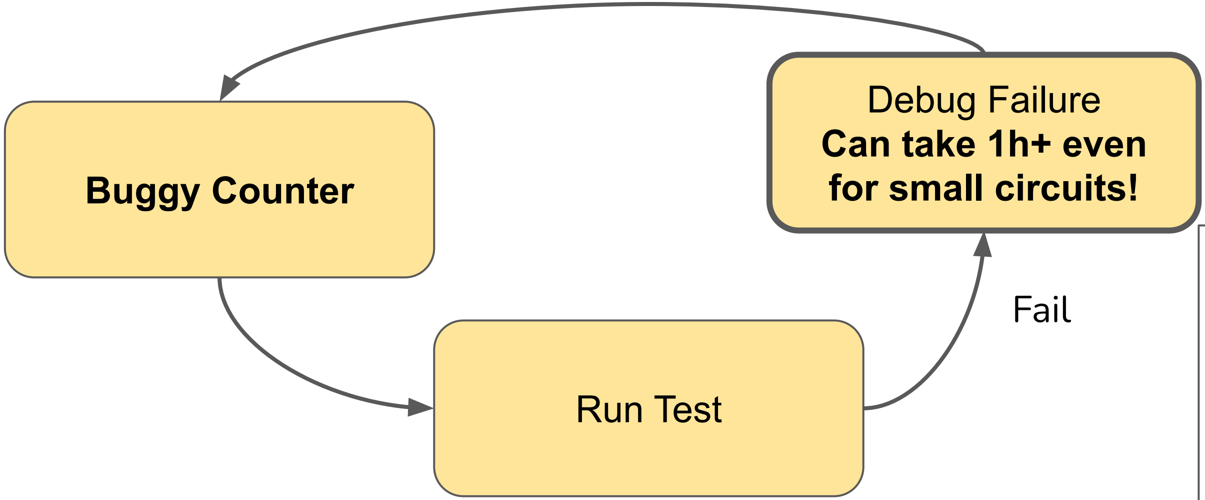
```
class Counter extends Module {
 val io = IO(new CounterIO)

 val count = RegInit(0.U(4.W))
 val overflow = RegInit(false.B)
 when(io.enable) {
   count := count + 1.U
 }
 when(count === "b0111".U) {
   overflow := true.B
 }

 io.count := count
 io.overflow := overflow
}
```

Automated RTL Repair



```
class Counter extends Module {
 val io = IO(new CounterIO)

 val count = RegInit(0.U(4.W))
 val overflow = RegInit(false.B)
 when(io.enable) {
   count := count + 1.U
 }
 when(count === "b0111".U) {
   overflow := true.B
 }

 io.count := count
 io.overflow := overflow
}
```

Automated RTL Repair



Buggy Counter

Run Test

Fail

Debug Failure
Can take 1h+ even for small circuits!

```
class Counter extends Module {
 val io = IO(new CounterIO)

 val count = RegInit(0.U(4.W))
 val overflow = RegInit(false.B)
 when(io.enable) {
   count := count + 1.U
 }
 when(count === "b0111".U) {
   overflow := true.B
 }

 io.count := count
 io.overflow := overflow
}
```

“Have you tried replacing b0111 with b1111”?

Automated RTL Repair

Problem Statement

- **given** a RTL design written in Verilog

Automated RTL Repair

Problem Statement

- **given** a RTL design written in Verilog
- **given** a failing input / output trace

Automated RTL Repair

Problem Statement

- **given** a RTL design written in Verilog
- **given** a failing input / output trace
- can we **find a (minimal) change** to the Verilog code that will make the **input / output trace pass**?

Prior Work: CirFix[1]

- provides a benchmark consisting of 32 defects across 11 different designs

[1]: Ahmad, Hammad, Yu Huang, and Westley Weimer.
"CirFix: Automatically Repairing Defects in Hardware Design Code."
27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, 2022.

[2]: Le Goues, Claire, ThanhVu Nguyen, Stephanie Forrest, and Westley Weimer.
"GenProg: A Generic Method for Automatic Software Repair."
IEEE Transactions on Software Engineering 2011

Prior Work: CirFix[1]

- provides a benchmark consisting of 32 defects across 11 different designs
- proposes a **genetic algorithm based** technique modelled after GenProg[2] which uses **repair templates and mutation** to find a possible repair

[1]: Ahmad, Hammad, Yu Huang, and Westley Weimer.
"CirFix: Automatically Repairing Defects in Hardware Design Code."
27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, 2022.

[2]: Le Goues, Claire, ThanhVu Nguyen, Stephanie Forrest, and Westley Weimer.
"GenProg: A Generic Method for Automatic Software Repair."
IEEE Transactions on Software Engineering 2011

Prior Work: CirFix[1]

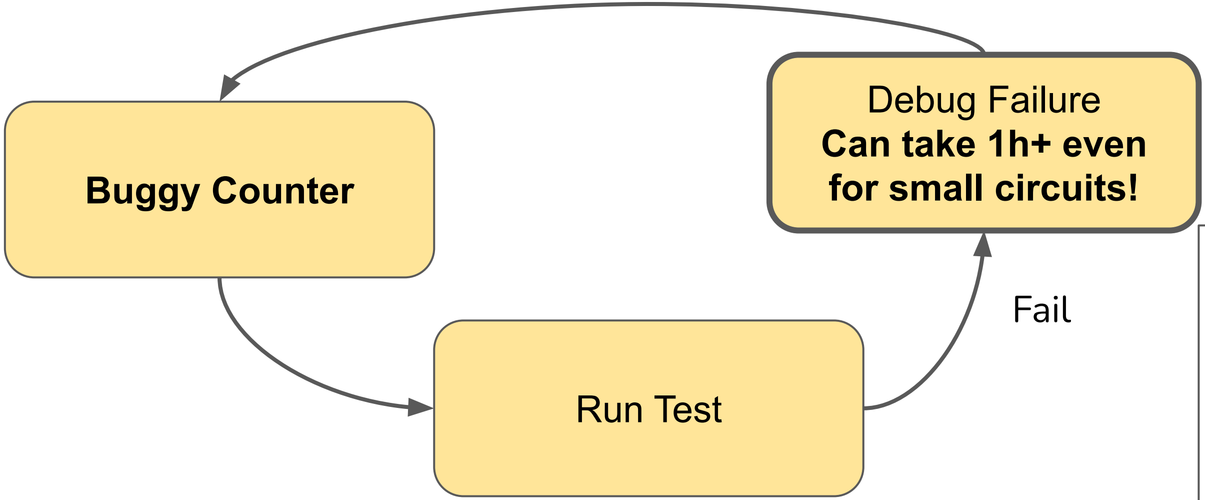
- provides a benchmark consisting of 32 defects across 11 different designs
- proposes a **genetic algorithm based** technique modelled after GenProg[2] which uses **repair templates and mutation** to find a possible repair
- repairs can take from **8s - 8h**

[1]: Ahmad, Hammad, Yu Huang, and Westley Weimer.
"CirFix: Automatically Repairing Defects in Hardware Design Code."
27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, 2022.

[2]: Le Goues, Claire, ThanhVu Nguyen, Stephanie Forrest, and Westley Weimer.
"GenProg: A Generic Method for Automatic Software Repair."
IEEE Transactions on Software Engineering 2011

Automated RTL Repair

8s - 8h is too long to wait!



Buggy Counter

Run Test

Fail

Debug Failure
Can take 1h+ even for small circuits!

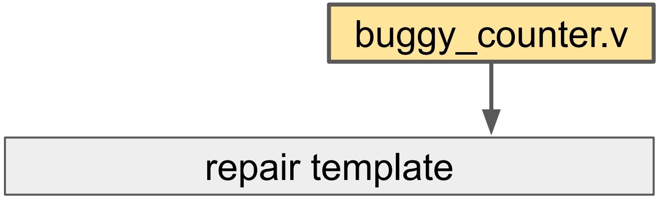
```
class Counter extends Module {
 val io = IO(new CounterIO)

 val count = RegInit(0.U(4.W))
 val overflow = RegInit(false.B)
 when(io.enable) {
   count := count + 1.U
 }
 when(count === "b0111".U) {
   overflow := true.B
 }

 io.count := count
 io.overflow := overflow
}
```

"Have you tried replacing b0111 with b1111"?

CirFix Repair Templates

buggy_counter.v

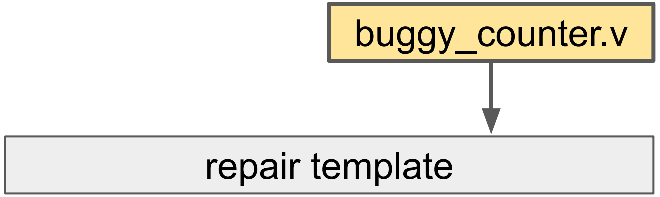
CirFix Repair Templates



buggy_counter.v

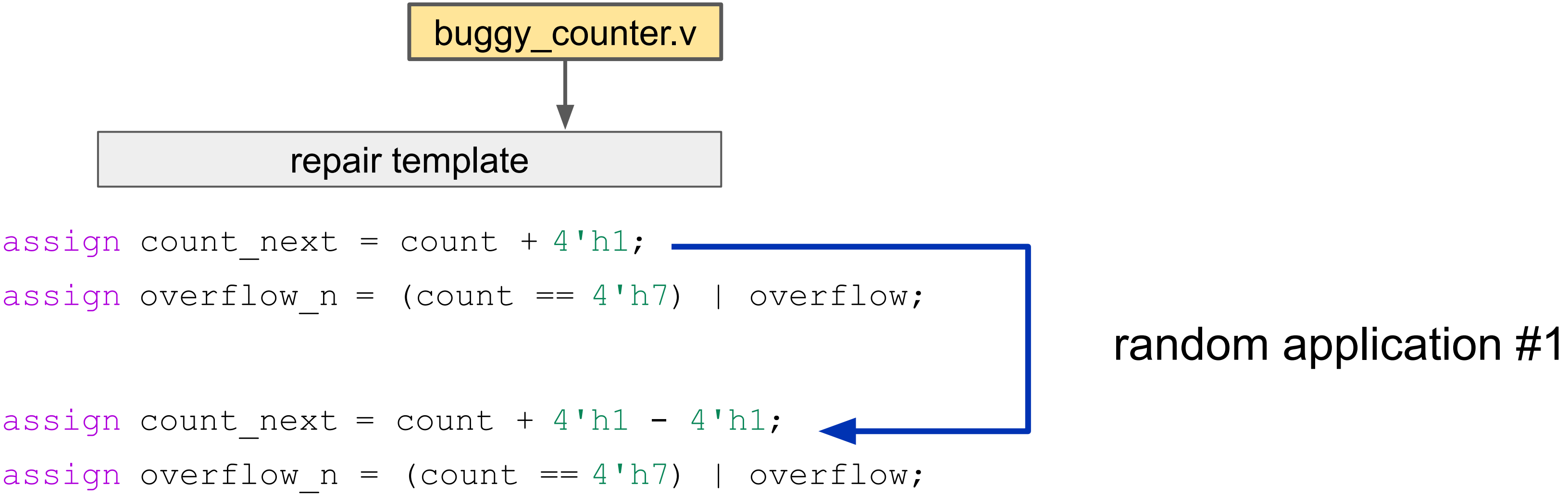
repair template

CirFix Repair Templates



```
assign count_next = count + 4'h1;
assign overflow_n = (count == 4'h7) | overflow;
```

CirFix Repair Templates



buggy_counter.v

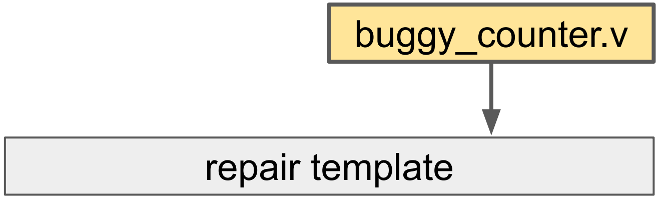
repair template

```
assign count_next = count + 4'h1;
assign overflow_n = (count == 4'h7) | overflow;
```

random application #1

```
assign count_next = count + 4'h1 - 4'h1;
assign overflow_n = (count == 4'h7) | overflow;
```

CirFix Repair Templates

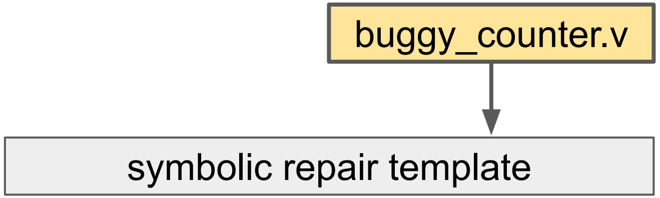


```
assign count_next = count + 4'h1;
assign overflow_n = (count == 4'h7) | overflow;
```

random application #2

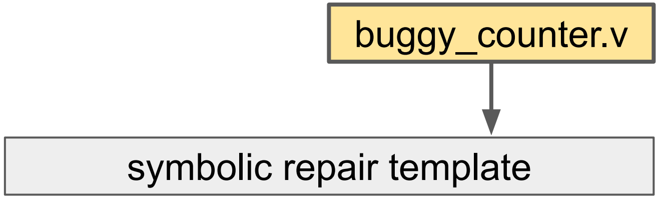
```
assign count_next = count + 4'h1;
assign overflow_n = (count == (4'h7 - 4'h1)) | overflow;
```

Formal Verification to the Rescue!



```
assign count_next = count + 4'h1;
assign overflow_n = (count == 4'h7) | overflow;
```

Formal Verification to the Rescue!

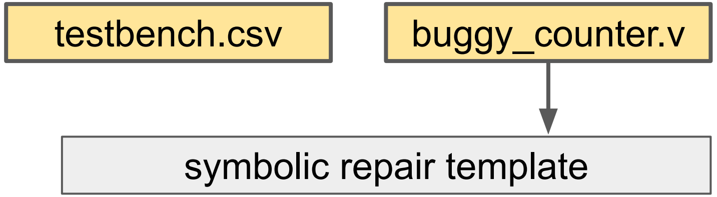


Expresses **all possible changes** guarded by synthesis variables.

```
assign count_next = count + 4'h1;
assign overflow_n = (count == 4'h7) | overflow;
```

```
assign count_next = count + ((φ_0)? α_0 : 4'h1);
assign overflow_n = (count == ((φ_1)? α_1 : 4'h7)) | overflow;
```

Formal Verification to the Rescue!



testbench.csv

buggy_counter.v

symbolic repair template

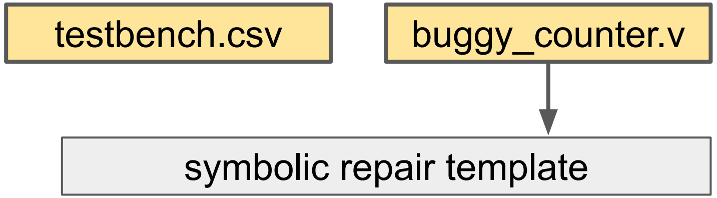
```
assign count_next = count + 4'h1;
assign overflow_n = (count == 4'h7) | overflow;


assign count_next = count + ((φ_0)? α_0 : 4'h1);
assign overflow_n = (count == ((φ_1)? α_1 : 4'h7)) | overflow;
```

<u>Standard Bounded Model Checking:</u>

∃ enable$_0$, enable$_1$. error

(find inputs such that an assertion is violated)

Formal Verification to the Rescue!



```
assign count_next = count + 4'h1;
assign overflow_n = (count == 4'h7) | overflow;
```

Standard Bounded Model Checking:

$\exists$ enable$_0$, enable$_1$. error

(find inputs such that an assertion is violated)

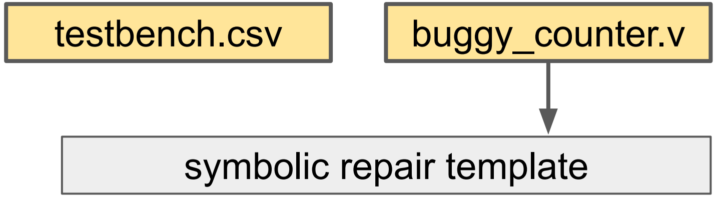
Repair Query

$\exists$ φ_i, α_i . enable$_0$ = 1 $\wedge$ count$_0$ = 0 $\wedge$ ....

(find synthesis constants such that inputs and outputs conform to our testbench trace)

```
assign count_next = count + ((φ_0)? α_0 : 4'h1);
assign overflow_n = (count == ((φ_1)? α_1 : 4'h7)) | overflow;
```

Formal Verification to the Rescue!



```
assign count_next = count + 4'h1;
assign overflow_n = (count == 4'h7) | overflow;
```

```
assign count_next = count + ((φ0)? α0  : 4'h1);
assign overflow_n = (count == ((φ1)? α1  : 4'h7)) | overflow;
```

<u>Standard Bounded Model Checking:</u>

$\exists$ enable$_0$, enable$_1$. error

(find inputs such that an assertion is violated)

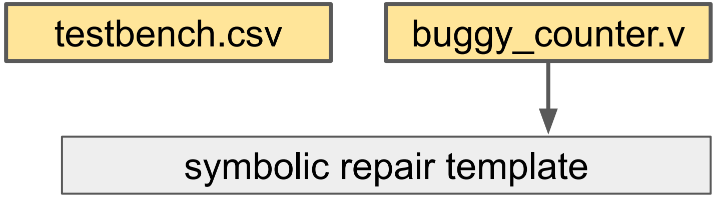
<u>Repair Query</u>

$\exists\ \varphi_i, \alpha_i\ .\ \text{enable}_0 = 1 \land \text{count}_0 = 0 \land$ ....

(find synthesis constants such that inputs and outputs conform to our testbench trace)

minimal change: s.t. min(sum(φ_i))

Formal Verification to the Rescue!



```
assign count_next = count + 4'h1;
assign overflow_n = (count == 4'h7) | overflow;
```

<u>Standard Bounded Model Checking:</u>

$\exists$ enable$_0$, enable$_1$. error

(find inputs such that an assertion is violated)

<u>Repair Query</u>

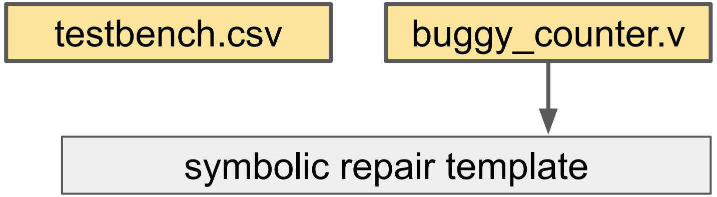
$\exists$ φ_i, α_i . enable$_0 = 1 \wedge$ count$_0 = 0 \wedge$ ....

(find synthesis constants such that inputs and outputs conform to our testbench trace)

```
assign count_next = count + ((φ_0)? α_0  : 4'h1);
assign overflow_n = (count == ((φ_1)? α_1  : 4'h7)) | overflow;
```

minimal change: s.t. min(sum(φ_i))   assert(true   ) → SAT / UNSAT

Formal Verification to the Rescue!



testbench.csv

buggy_counter.v

symbolic repair template

```
assign count_next = count + 4'h1;
assign overflow_n = (count == 4'h7) | overflow;
```

```
assign count_next = count + ((φ_0)? α_0 : 4'h1);
assign overflow_n = (count == ((φ_1)? α_1 : 4'h7)) | overflow;
```

Standard Bounded Model Checking:

$\exists$ $enable_0, enable_1 .$ error

(find inputs such that an assertion is violated)

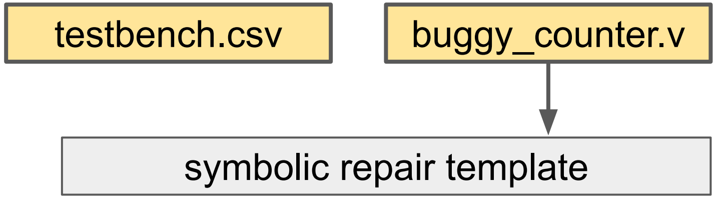
Repair Query

$\exists$ $\varphi_i, \alpha_i . enable_0 = 1 \land count_0 = 0 \land ....$

(find synthesis constants such that inputs and outputs conform to our testbench trace)

minimal change: s.t. min(sum(φ_i))   assert(true   ) → $\boxed{\text{SAT}}$ / UNSAT

Formal Verification to the Rescue!



```
assign count_next = count + 4'h1;
assign overflow_n = (count == 4'h7) | overflow;
```

Standard Bounded Model Checking:

$\exists$ $\text{enable}_0, \text{enable}_1$. error

(find inputs such that an assertion is violated)

Repair Query

$\exists$ φ_i, α_i . $\text{enable}_0 = 1 \land \text{count}_0 = 0 \land$ ....

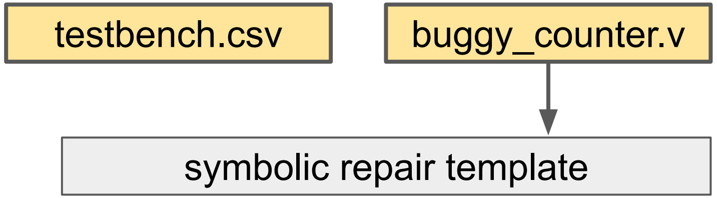
(find synthesis constants such that inputs and outputs conform to our testbench trace)

```
assign count_next = count + ((φ_0)? α_0  : 4'h1);
assign overflow_n = (count == ((φ_1)? α_1  : 4'h7)) | overflow;
```

minimal change: s.t. min(sum(φ_i))

assert(sum(φ_i) == 1) → SAT / $\boxed{\text{UNSAT}}$

Formal Verification to the Rescue!



```
assign count_next = count + 4'h1;
assign overflow_n = (count == 4'h7) | overflow;
```

```
assign count_next = count + ((φ_0)? α_0 : 4'h1);
assign overflow_n = (count == ((φ_1)? α_1 : 4'h7)) | overflow;
```

Standard Bounded Model Checking:

$\exists$ $enable_0$, $enable_1$. error

(find inputs such that an assertion is violated)

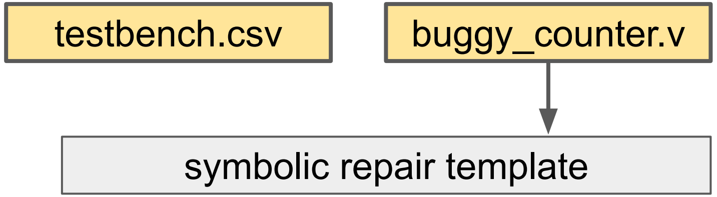
Repair Query

$\exists$ φ_i, α_i . $enable_0 = 1 \land count_0 = 0 \land$ ....

(find synthesis constants such that inputs and outputs conform to our testbench trace)

minimal change: s.t. min(sum(φ_i))   assert(sum(φ_i) == 2) → $\boxed{\text{SAT}}$ / UNSAT

Formal Verification to the Rescue!



testbench.csv

buggy_counter.v

symbolic repair template

```
assign count_next = count + 4'h1;
assign overflow_n = (count == 4'h7) | overflow;
```

<u>Standard Bounded Model Checking:</u>

$\exists\ \text{enable}_0, \text{enable}_1\ .\ \text{error}$

(find inputs such that an assertion is violated)

<u>Repair Query</u>

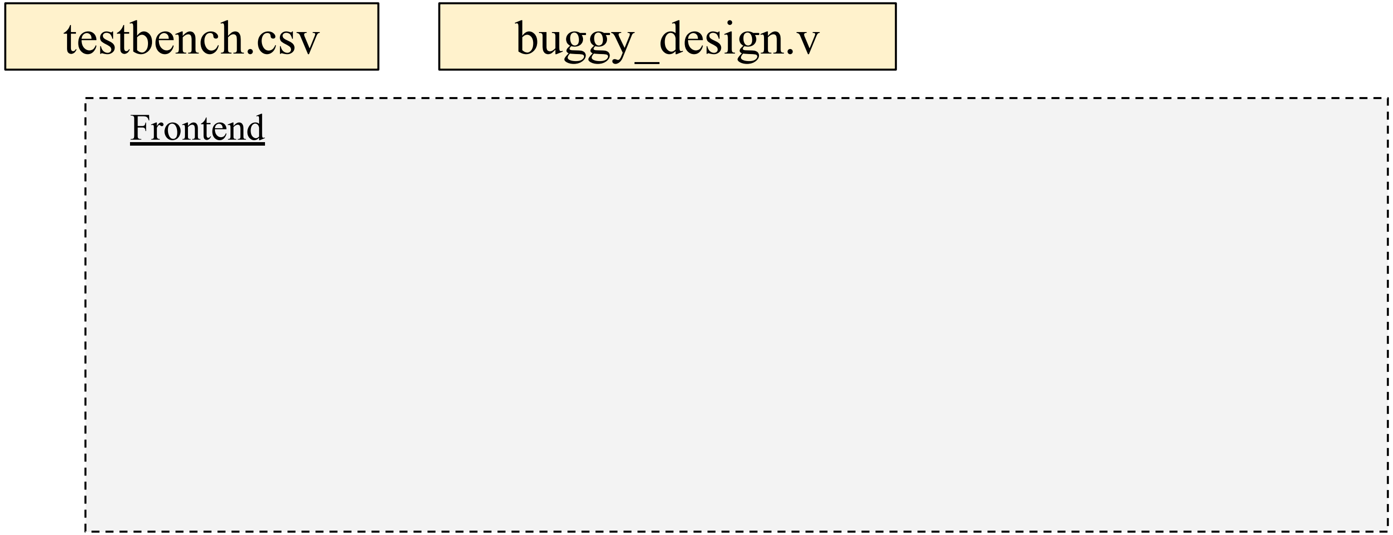
$\exists\ \varphi_i, \alpha_i\ .\ \text{enable}_0 = 1 \land \text{count}_0 = 0 \land \ldots.$

(find synthesis constants such that inputs and outputs conform to our testbench trace)

```
assign count_next = count + ((φ_0)? α_0  : 4'h1);
assign overflow_n = (count == ((φ_1)? α_1  : 4'h7)) | overflow;
```

minimal change: s.t. min(sum(φ_i)) assert(sum(φ_i) == 2) → $\boxed{\text{SAT}}$ / UNSAT ✓

System Overview

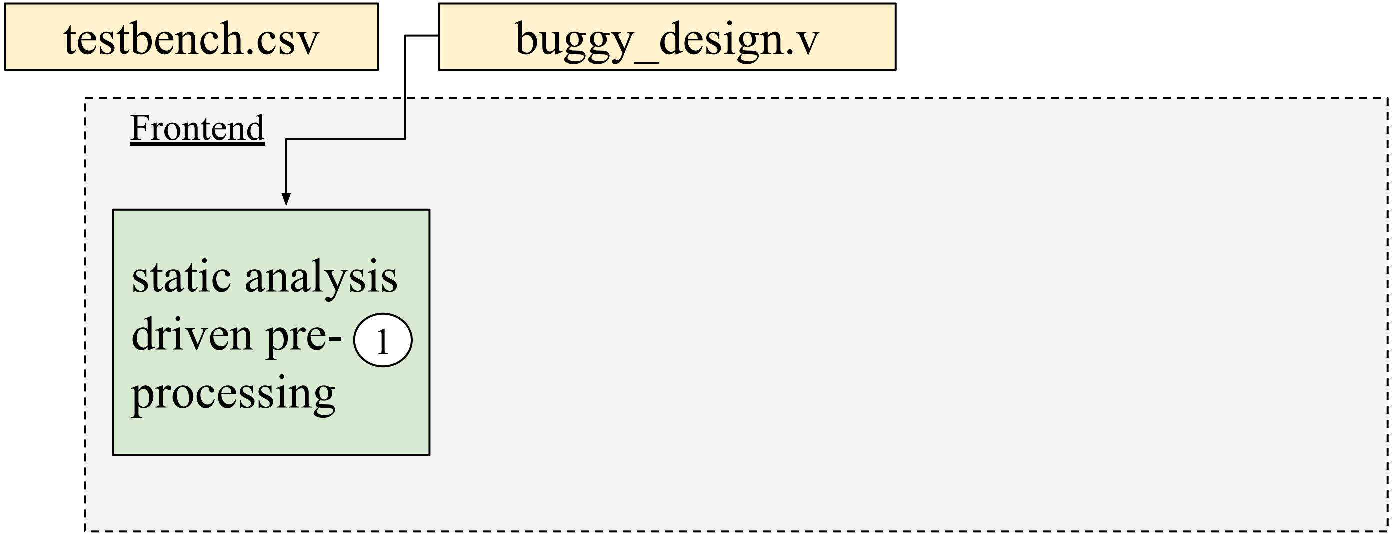


testbench.csv

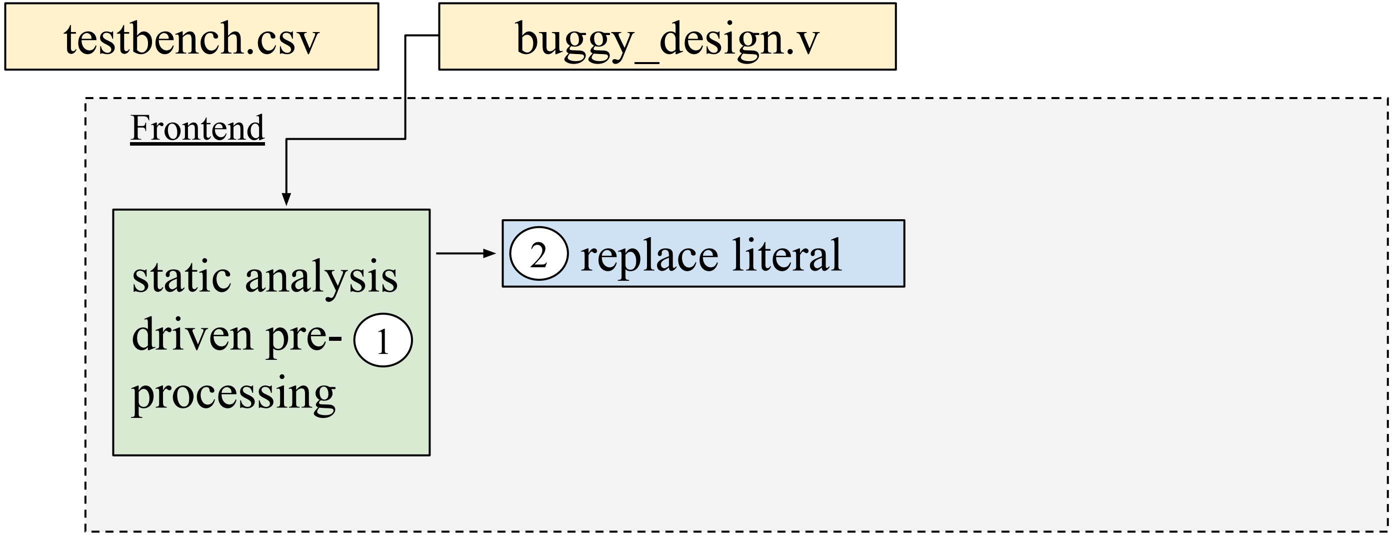
buggy_design.v

Frontend

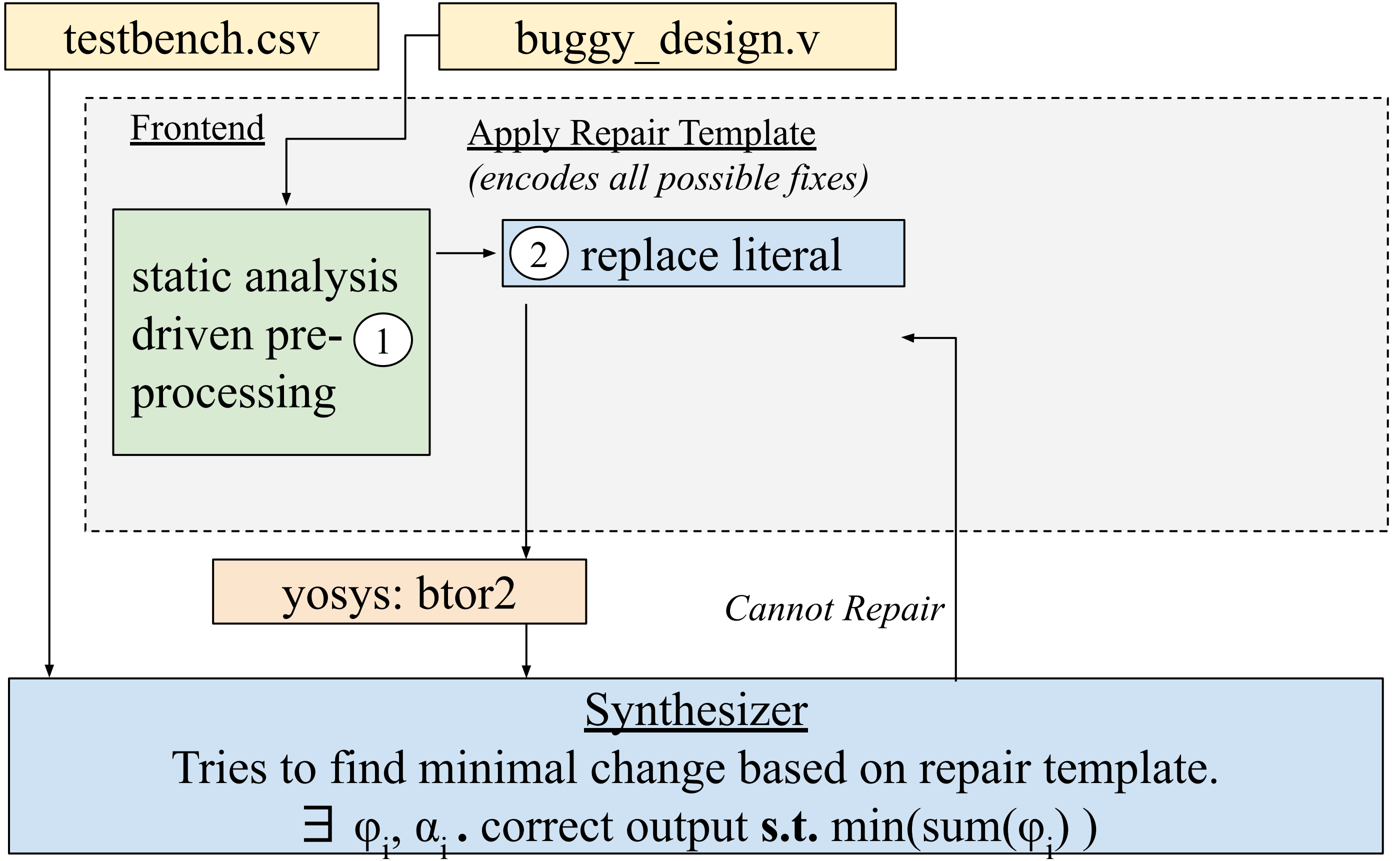
System Overview



System Overview



System Overview



testbench.csv

buggy_design.v

Frontend

Apply Repair Template

(encodes all possible fixes)

static analysis driven pre-processing ①

② replace literal

yosys: btor2

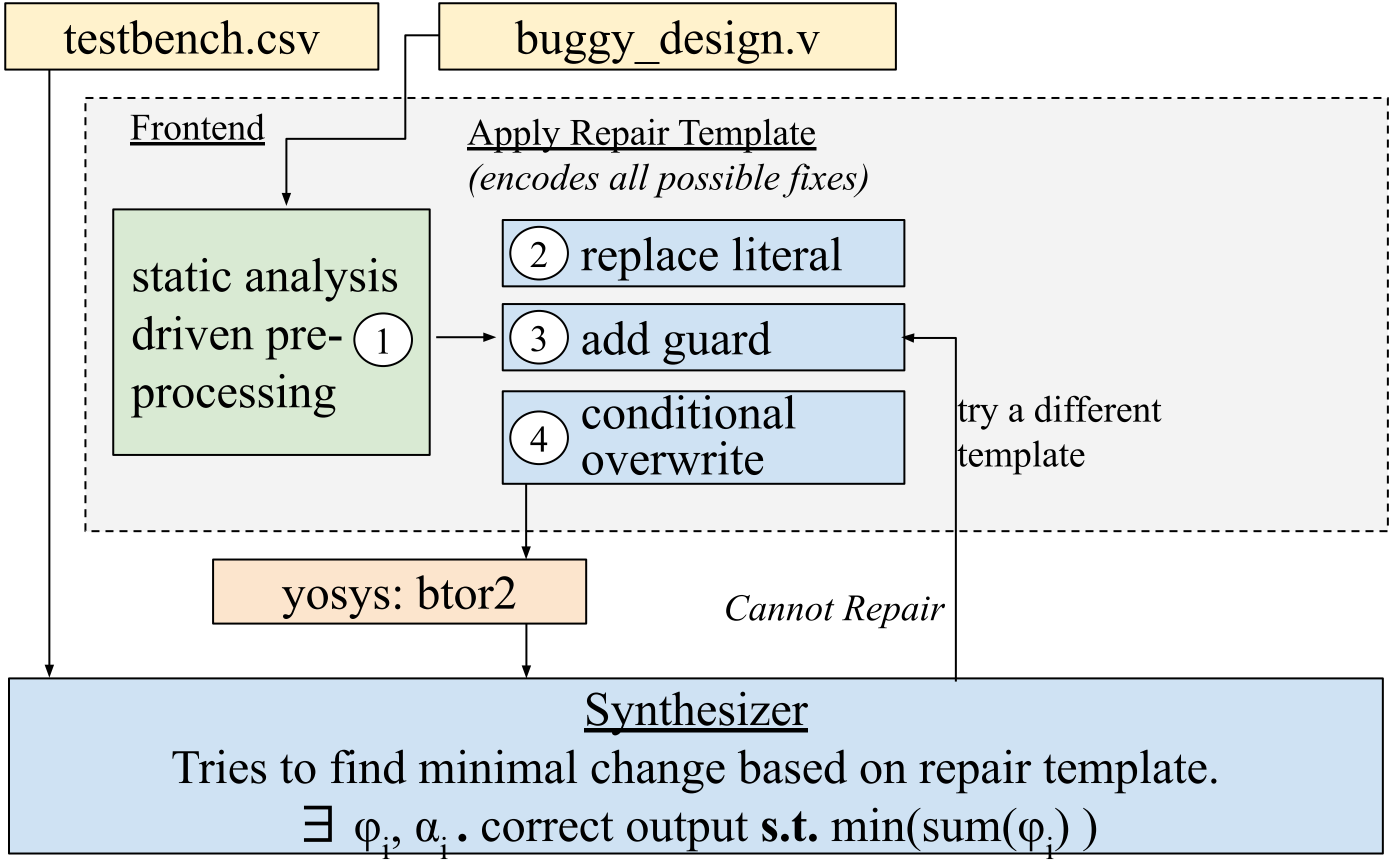
Cannot Repair

Synthesizer

Tries to find minimal change based on repair template.

$\exists\ \varphi_i, \alpha_i$. correct output **s.t.** $\min(\text{sum}(\varphi_i))$

System Overview



testbench.csv

buggy_design.v

Frontend

Apply Repair Template

(encodes all possible fixes)

static analysis driven pre-processing ①

② replace literal

③ add guard

④ conditional overwrite

try a different template

yosys: btor2

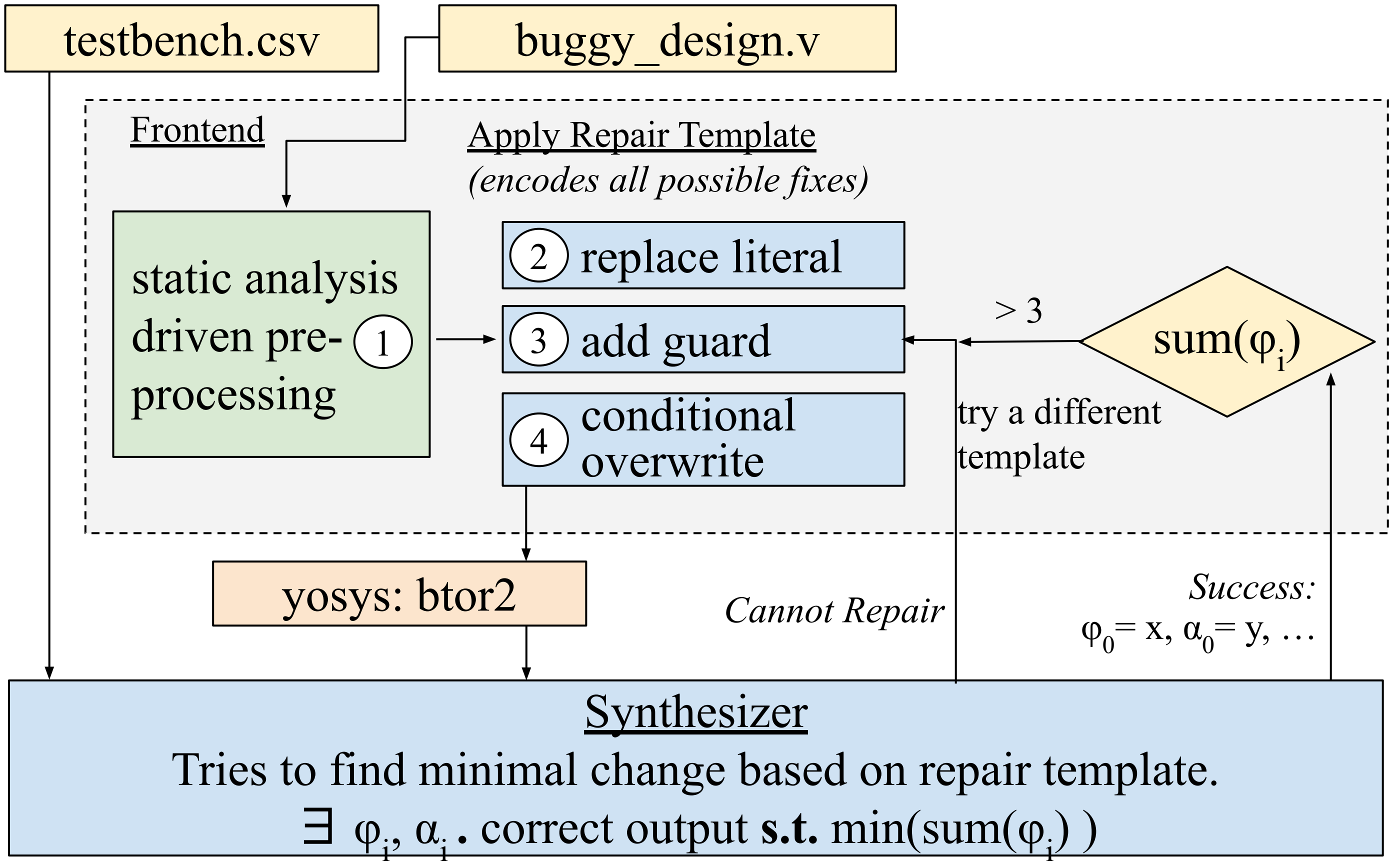
Cannot Repair

Synthesizer

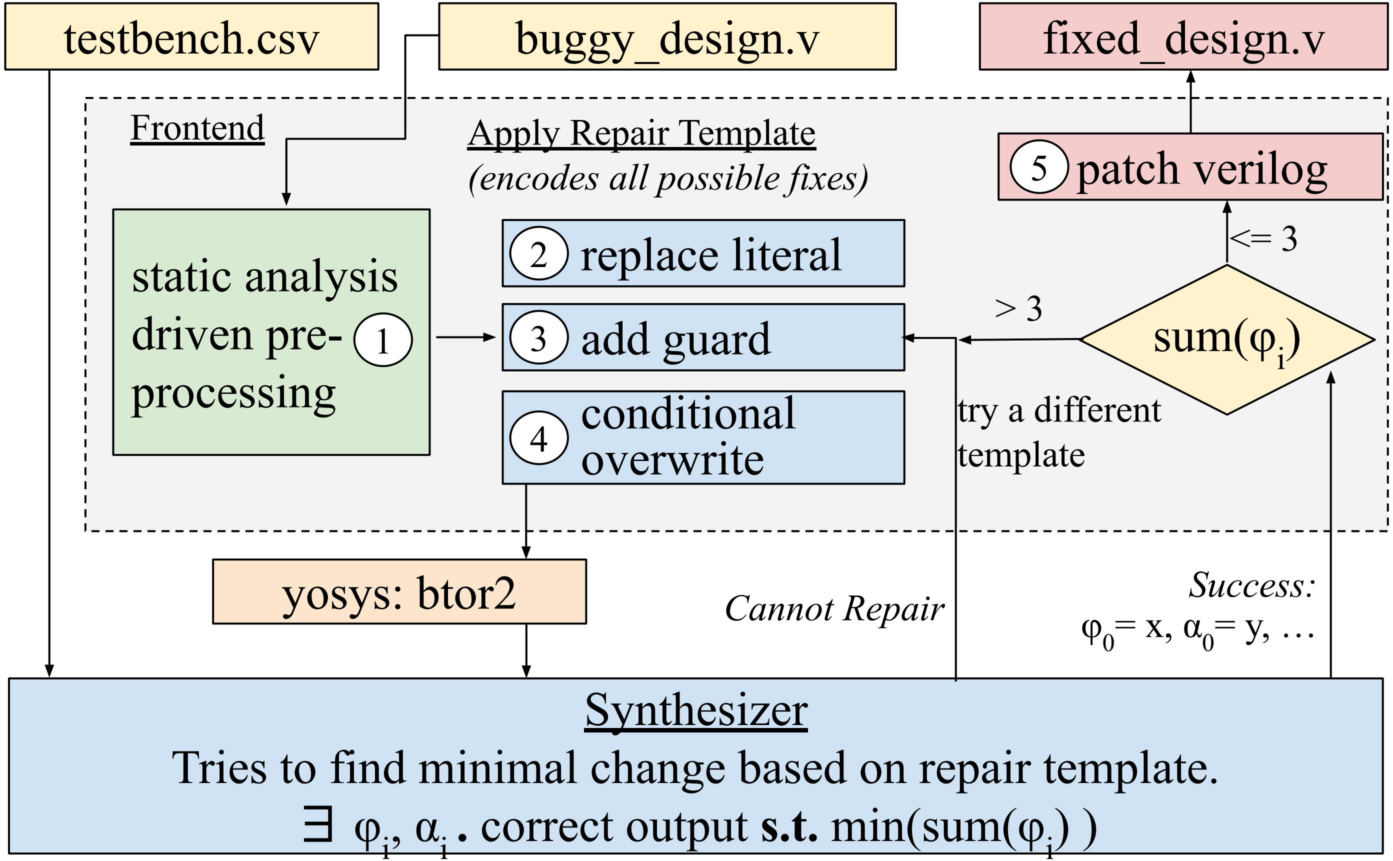
Tries to find minimal change based on repair template.

$\exists\ \varphi_i, \alpha_i$. correct output **s.t.** min(sum(φ_i))

System Overview


testbench.csv
buggy_design.v
Frontend
Apply Repair Template
(encodes all possible fixes)
static analysis driven pre-processing
1
2 replace literal
3 add guard
4 conditional overwrite
> 3
sum(φ_i)
try a different template
yosys: btor2
Cannot Repair
Success:
φ_0= x, α_0= y, …
Synthesizer
Tries to find minimal change based on repair template.
$\exists\ \varphi_i, \alpha_i$. correct output **s.t.** min(sum(φ_i))

System Overview



testbench.csv

buggy_design.v

fixed_design.v

Frontend

1 static analysis driven pre-processing

Apply Repair Template

(encodes all possible fixes)

2 replace literal

3 add guard

4 conditional overwrite

5 patch verilog

$\text{sum}(\varphi_i)$

<= 3

> 3

try a different template

yosys: btor2

Cannot Repair

Success:

$\varphi_0 = x,\ \alpha_0 = y,\ \ldots$

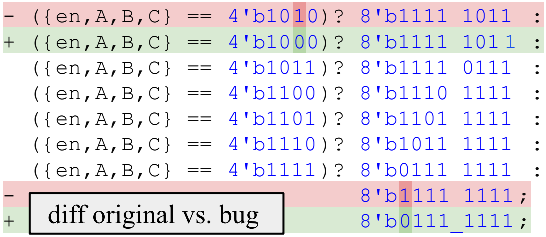
Synthesizer

Tries to find minimal change based on repair template.

$\exists\ \varphi_i, \alpha_i$. correct output **s.t.** $\min(\text{sum}(\varphi_i))$

Decoder Benchmark Repairs

decoder_w1: Two separate numeric errors



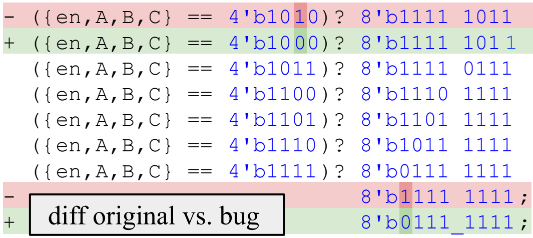
```
- ({en,A,B,C} == 4'b1010)? 8'b1111 1011  :
+ ({en,A,B,C} == 4'b1000)? 8'b1111 1011  :
  ({en,A,B,C} == 4'b1011)? 8'b1111 0111  :
  ({en,A,B,C} == 4'b1100)? 8'b1110 1111  :
  ({en,A,B,C} == 4'b1101)? 8'b1101 1111  :
  ({en,A,B,C} == 4'b1110)? 8'b1011 1111  :
  ({en,A,B,C} == 4'b1111)? 8'b0111 1111  :
-                          8'b1111 1111;
+                          8'b0111_1111;
```

diff original vs. bug

Our Tool: repair after 0.4s
Cirfix: repair after 7h (63,000x slower!)

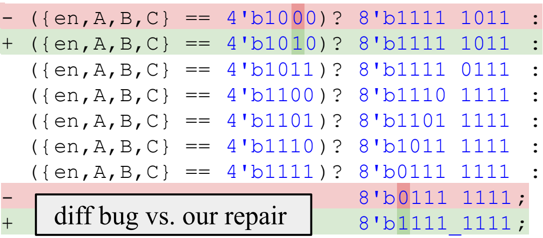
Decoder Benchmark Repairs

decoder_w1: Two separate numeric errors



```
- ({en,A,B,C} == 4'b1010)? 8'b1111 1011 :
+ ({en,A,B,C} == 4'b1000)? 8'b1111 1011 :
  ({en,A,B,C} == 4'b1011)? 8'b1111 0111 :
  ({en,A,B,C} == 4'b1100)? 8'b1110 1111 :
  ({en,A,B,C} == 4'b1101)? 8'b1101 1111 :
  ({en,A,B,C} == 4'b1110)? 8'b1011 1111 :
  ({en,A,B,C} == 4'b1111)? 8'b0111 1111 :
-                          8'b1111 1111;
+                          8'b0111_1111;
```

diff original vs. bug



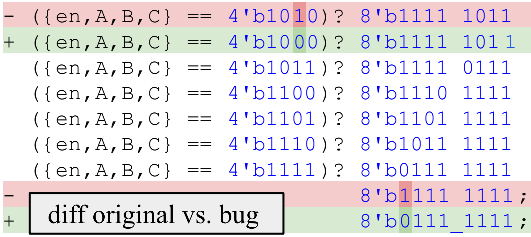
```
- ({en,A,B,C} == 4'b1000)? 8'b1111 1011 :
+ ({en,A,B,C} == 4'b1010)? 8'b1111 1011 :
  ({en,A,B,C} == 4'b1011)? 8'b1111 0111 :
  ({en,A,B,C} == 4'b1100)? 8'b1110 1111 :
  ({en,A,B,C} == 4'b1101)? 8'b1101 1111 :
  ({en,A,B,C} == 4'b1110)? 8'b1011 1111 :
  ({en,A,B,C} == 4'b1111)? 8'b0111 1111 :
-                          8'b0111 1111;
+                          8'b1111_1111;
```

diff bug vs. our repair

Our Tool: ✅ Correct repair after 0.4s
Cirfix: repair after 7h (63,000x slower!)

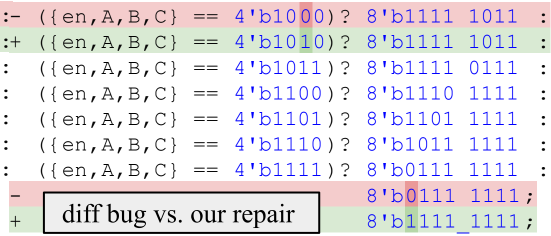
Decoder Benchmark Repairs

decoder_w1: Two separate numeric errors



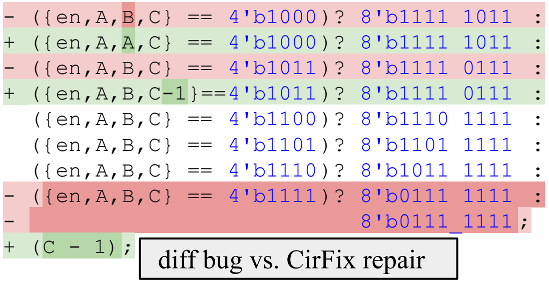
```
- ({en,A,B,C} == 4'b1010)? 8'b1111 1011 :
+ ({en,A,B,C} == 4'b1000)? 8'b1111 1011 :
  ({en,A,B,C} == 4'b1011)? 8'b1111 0111 :
  ({en,A,B,C} == 4'b1100)? 8'b1110 1111 :
  ({en,A,B,C} == 4'b1101)? 8'b1101 1111 :
  ({en,A,B,C} == 4'b1110)? 8'b1011 1111 :
  ({en,A,B,C} == 4'b1111)? 8'b0111 1111 :
-                          8'b1111 1111;
+                          8'b0111_1111;
```

diff original vs. bug



```
- ({en,A,B,C} == 4'b1000)? 8'b1111 1011 :
+ ({en,A,B,C} == 4'b1010)? 8'b1111 1011 :
  ({en,A,B,C} == 4'b1011)? 8'b1111 0111 :
  ({en,A,B,C} == 4'b1100)? 8'b1110 1111 :
  ({en,A,B,C} == 4'b1101)? 8'b1101 1111 :
  ({en,A,B,C} == 4'b1110)? 8'b1011 1111 :
  ({en,A,B,C} == 4'b1111)? 8'b0111 1111 :
-                          8'b0111 1111;
+                          8'b1111_1111;
```

diff bug vs. our repair



```
- ({en,A,B,C} == 4'b1000)? 8'b1111 1011 :
+ ({en,A,A,C} == 4'b1000)? 8'b1111 1011 :
- ({en,A,B,C} == 4'b1011)? 8'b1111 0111 :
+ ({en,A,B,C-1}==4'b1011)? 8'b1111 0111 :
  ({en,A,B,C} == 4'b1100)? 8'b1110 1111 :
  ({en,A,B,C} == 4'b1101)? 8'b1101 1111 :
  ({en,A,B,C} == 4'b1110)? 8'b1011 1111 :
- ({en,A,B,C} == 4'b1111)? 8'b0111 1111 :
-                          8'b0111_1111;
+ (C - 1);
```

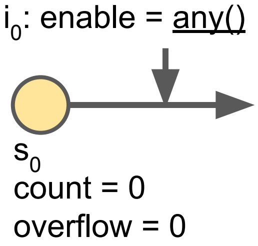
diff bug vs. CirFix repair

CirFix (7h): repair passes testbench, but changes code that is never tested.

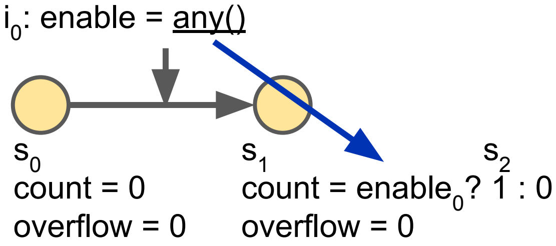
Out tool performs a *minimal* repair.

Our Tool: ✅ Correct repair after 0.4s
Cirfix: ❌ Incorrect repair after 7h (63,000x slower!)

Scalability Issues



Scalability Issues



i_0: enable = <u>any()</u>

s_0

count = 0

overflow = 0

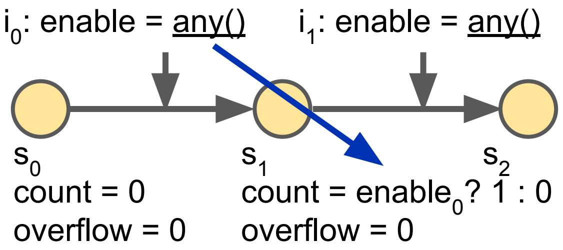
s_1

count = enable$_0$? 1 : 0

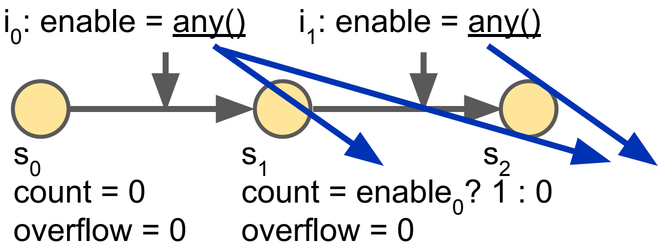
overflow = 0

s_2

Scalability Issues



Scalability Issues



i_0: enable = <u>any()</u>

i_1: enable = <u>any()</u>

s_0

count = 0

overflow = 0

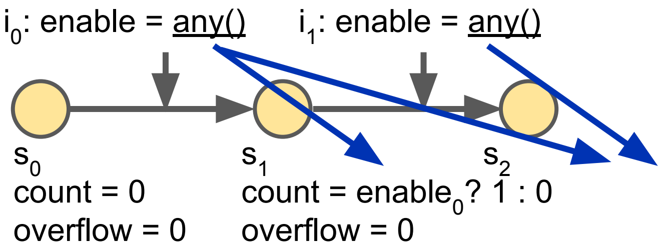
s_1

count = enable_0? 1 : 0

overflow = 0

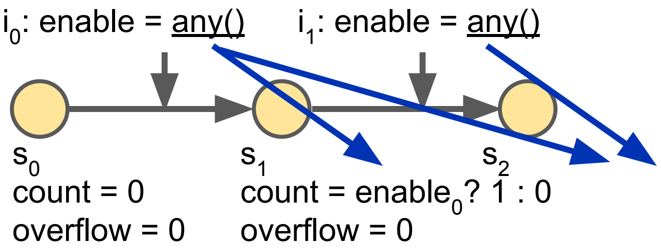
s_2

Scalability Issues



The more cycles we unroll for, the longer the solver takes!

Scalability Issues



i_0: enable = any()

i_1: enable = any()

s_0

count = 0

overflow = 0

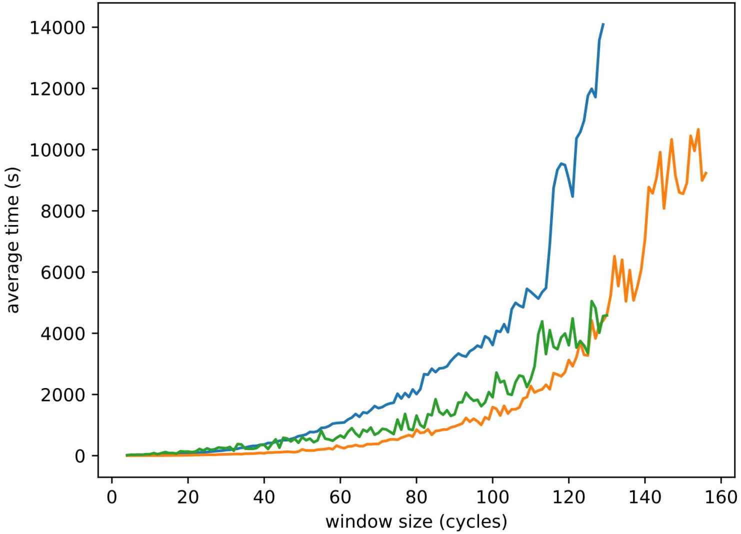
s_1

count = enable$_0$? 1 : 0

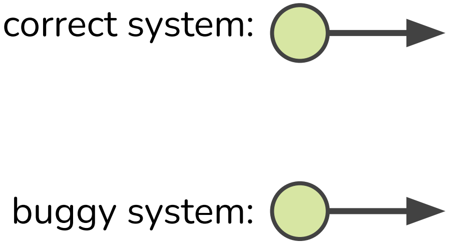
overflow = 0

s_2

The more cycles we unroll for, the longer the solver takes!



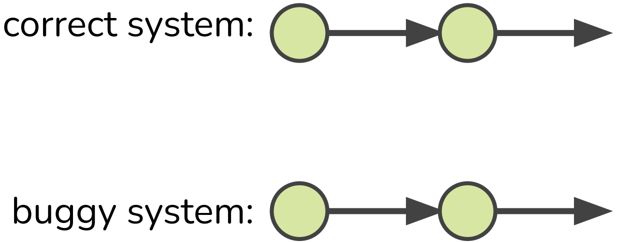
Test Execution



correct system:

buggy system:

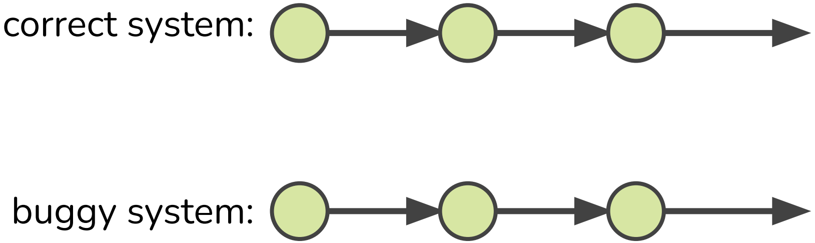
Test Execution



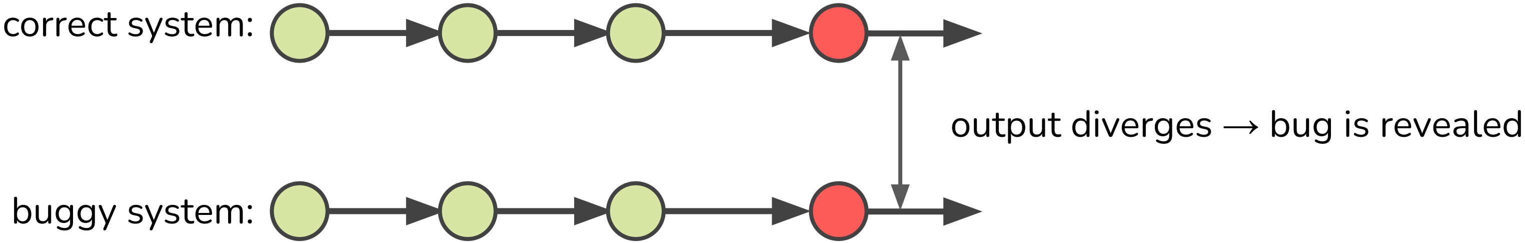
correct system:

buggy system:

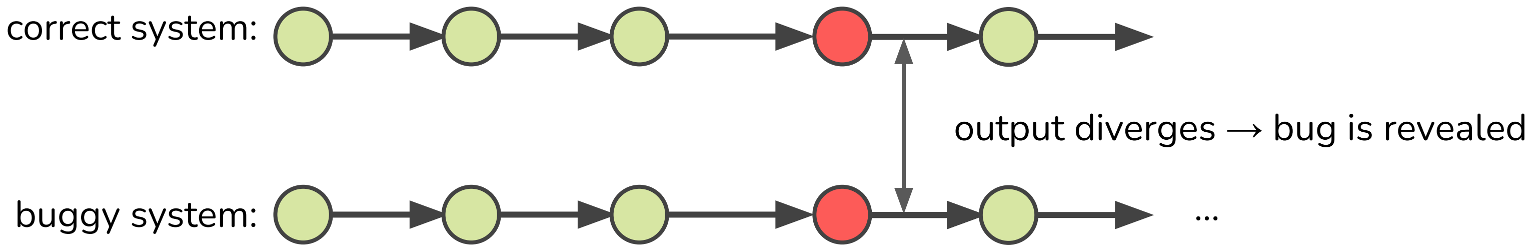
Test Execution



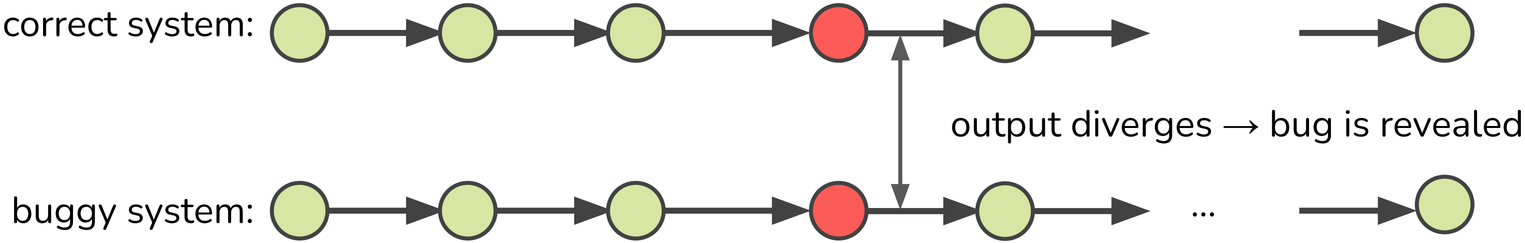
Test Execution



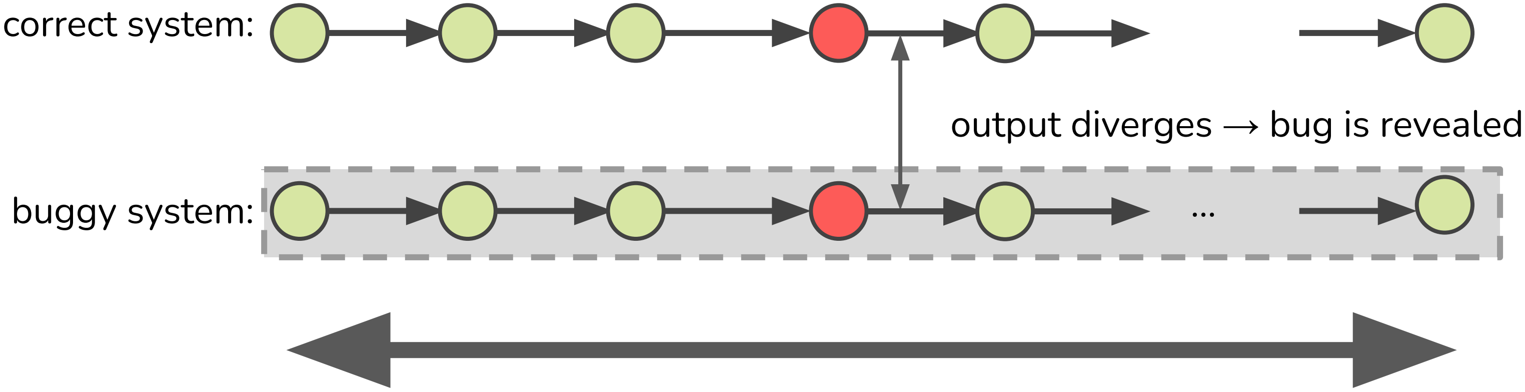
Test Execution



Test Execution

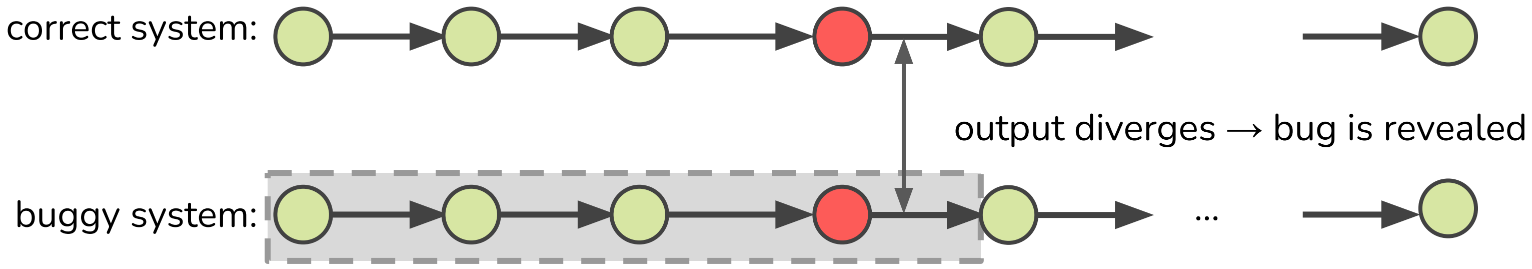


Test Execution

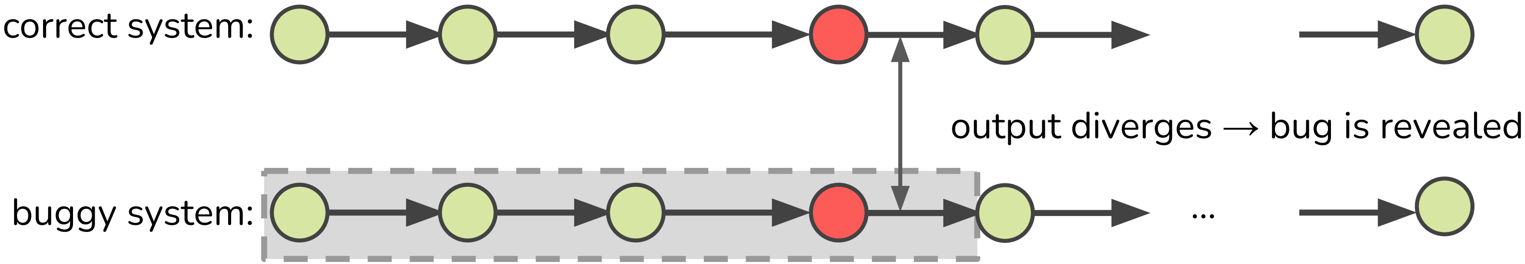


11 - 500k cycles

Only Symbolically Execute to First Failure

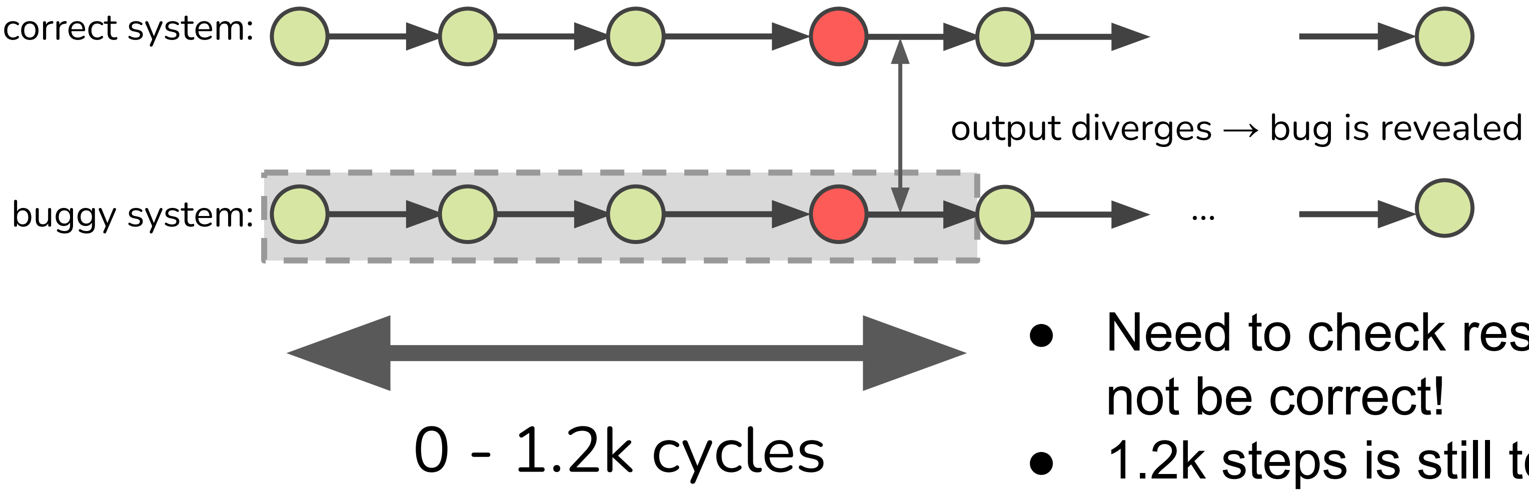


Only Symbolically Execute to First Failure



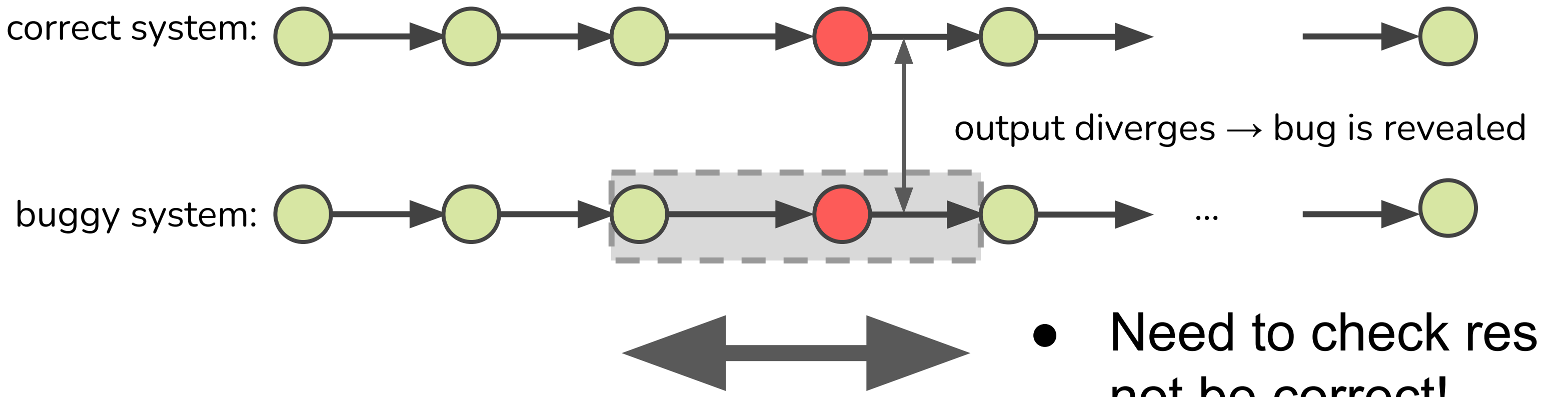
- Need to check result, may not be correct!

Only Symbolically Execute to First Failure



- Need to check result, may not be correct!
- 1.2k steps is still too much!

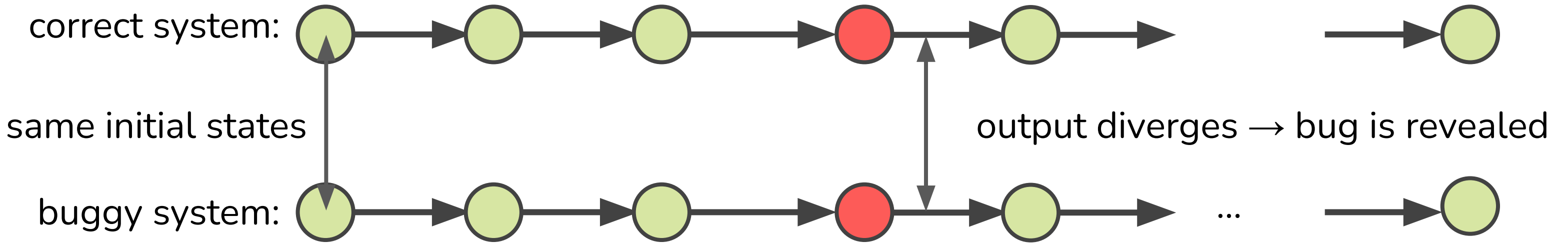
Adaptive Windowing



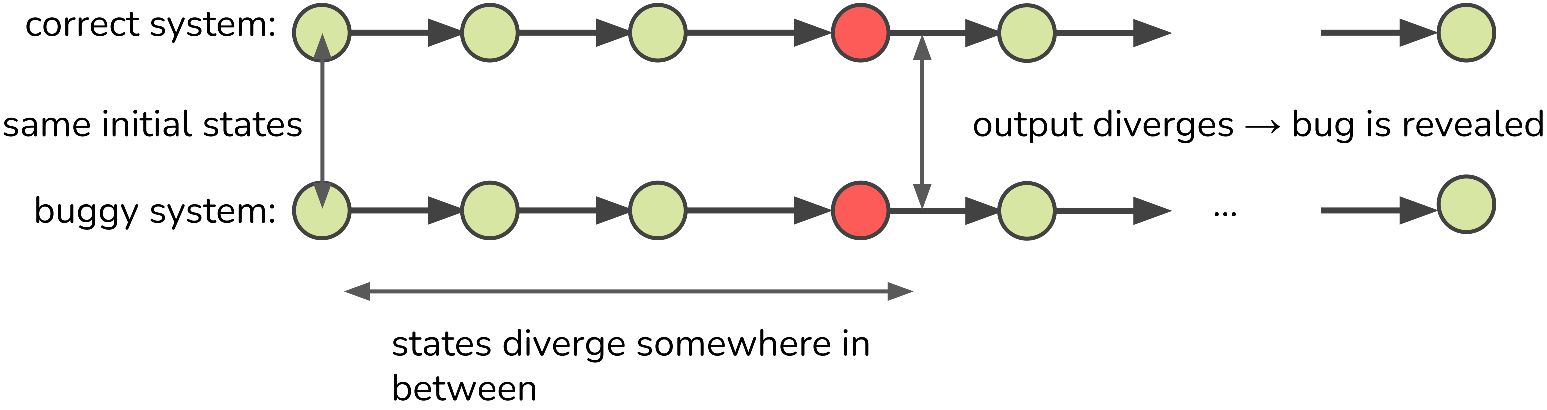
Can we make the window even smaller?

- Need to check result, may not be correct!
- 1.2k steps is still too much!

Adaptive Windowing

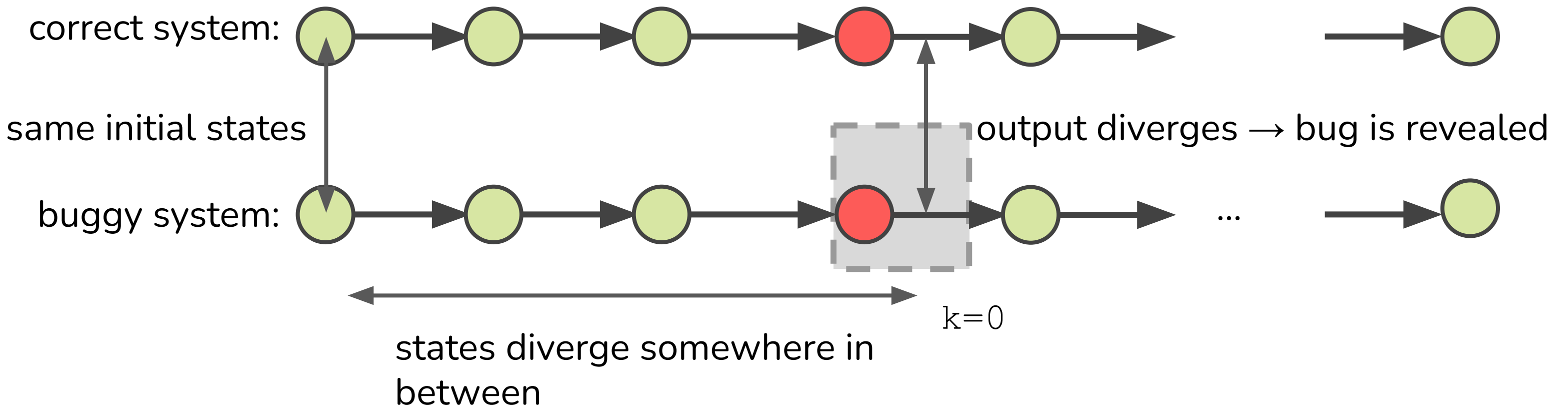


Adaptive Windowing



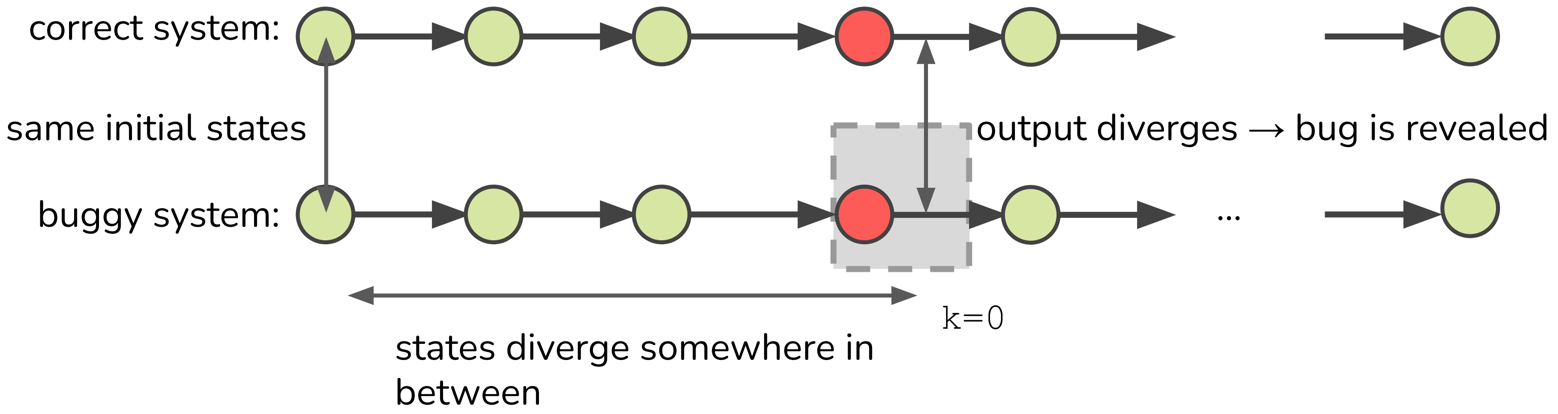
correct system:
same initial states
buggy system:
output diverges → bug is revealed
...
states diverge somewhere in between

Adaptive Windowing


correct system:
same initial states
buggy system:
output diverges → bug is revealed
...
k=0
states diverge somewhere in between

Adaptive Windowing

```
when(count === "b0111".U) {
  overflow := true.B
}
```



correct system:

same initial states

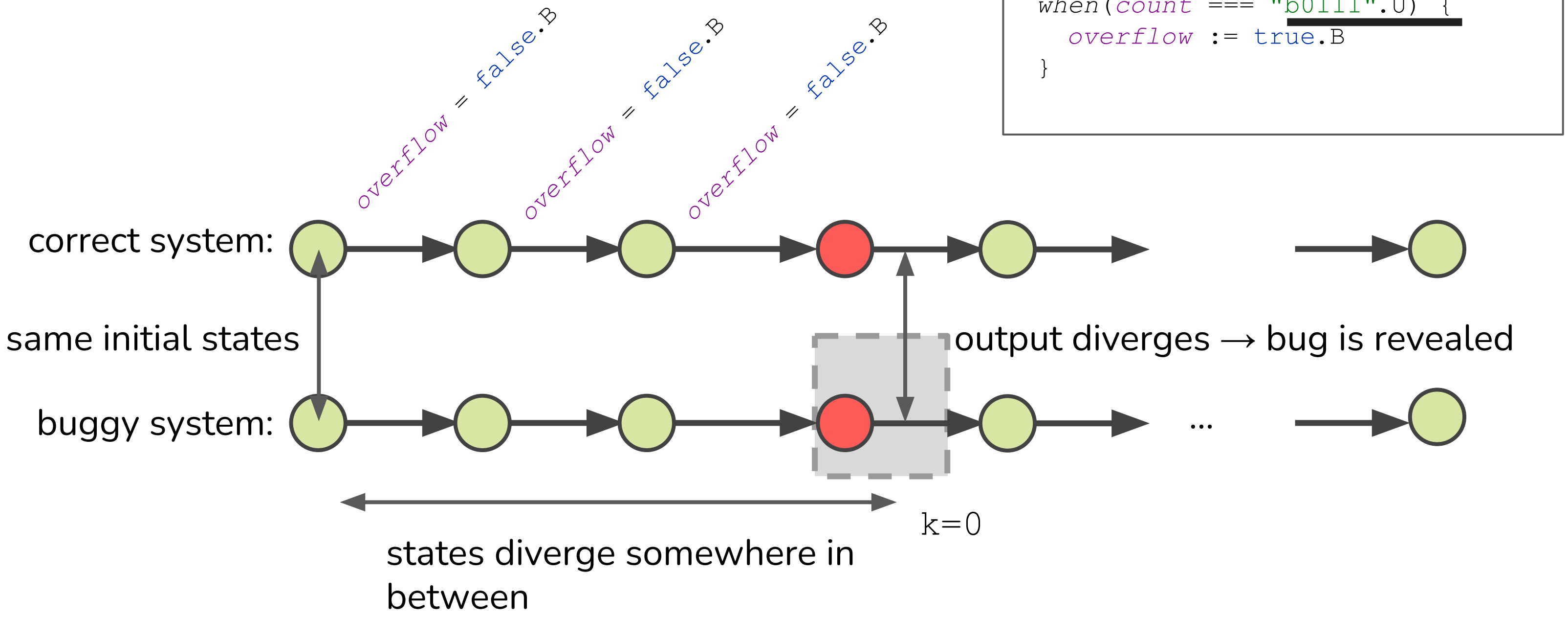
buggy system:

output diverges → bug is revealed

states diverge somewhere in between

Adaptive Windowing

```
when(count === "b0111".U) {
  overflow := true.B
}
```



overflow = false.B

overflow = false.B

overflow = false.B

correct system:

same initial states

buggy system:

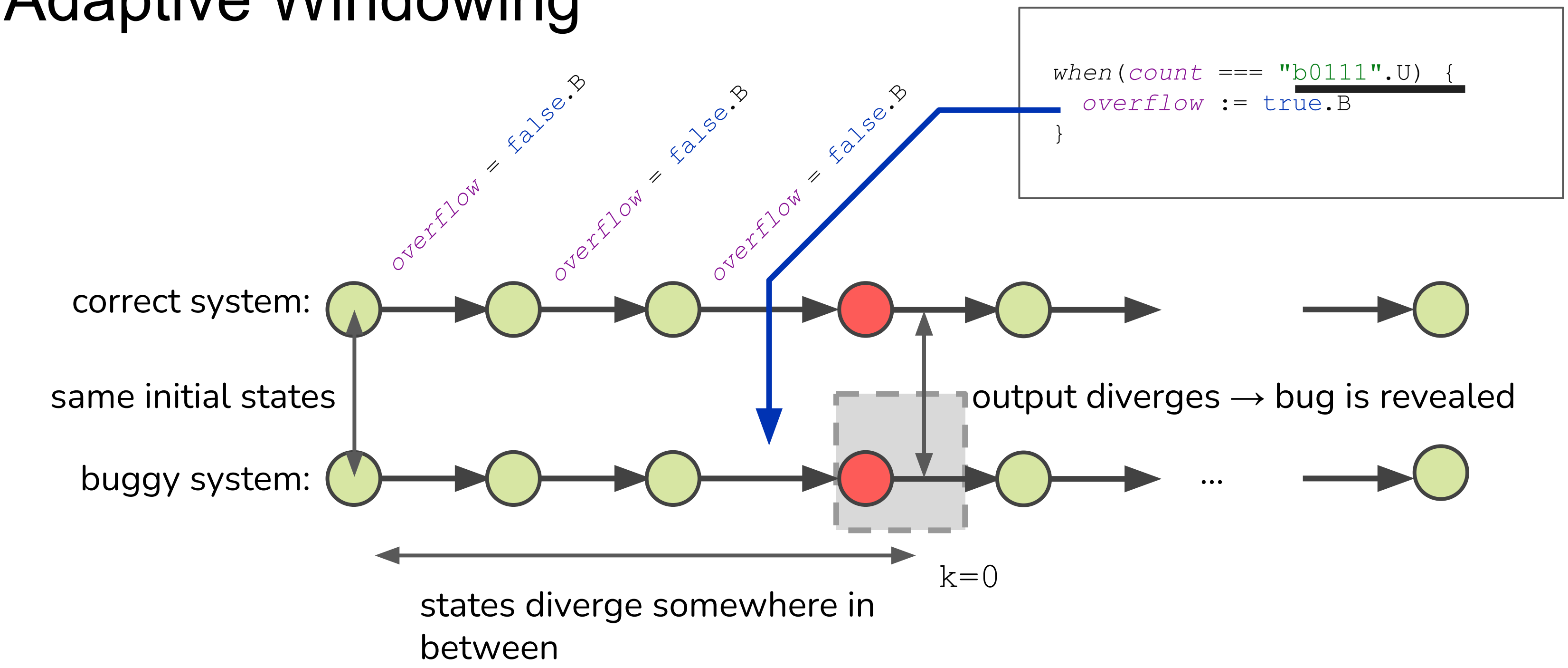
output diverges → bug is revealed

...

k=0

states diverge somewhere in between

Adaptive Windowing



```
when(count === "b0111".U) {
  overflow := true.B
}
```

correct system:

same initial states

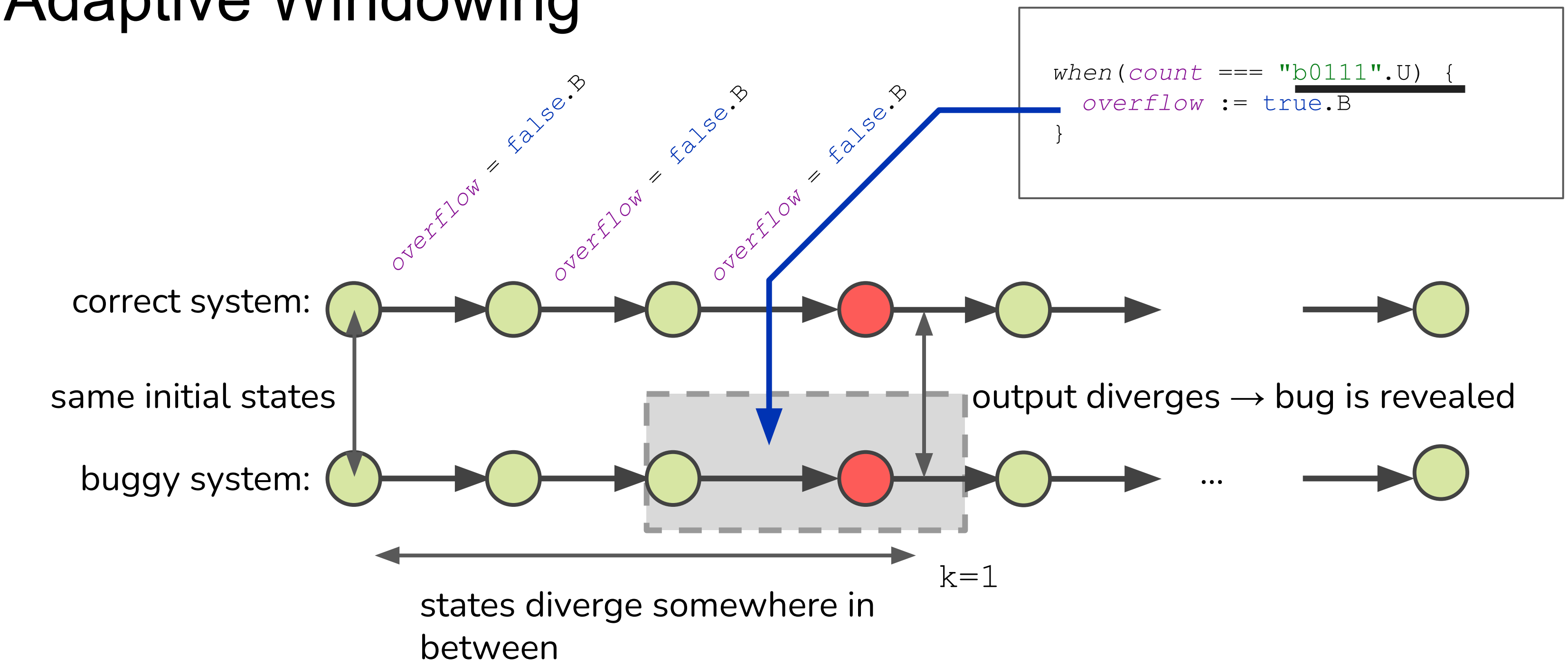
buggy system:

output diverges → bug is revealed

k=0

states diverge somewhere in between

Adaptive Windowing



```
when(count === "b0111".U) {
  overflow := true.B
}
```

correct system:

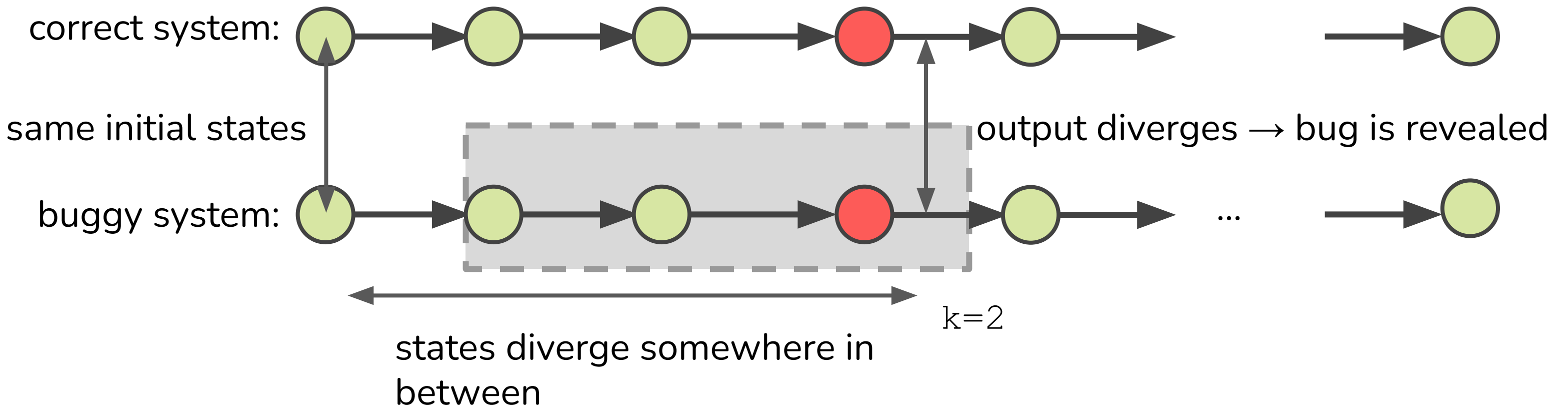
same initial states

buggy system:

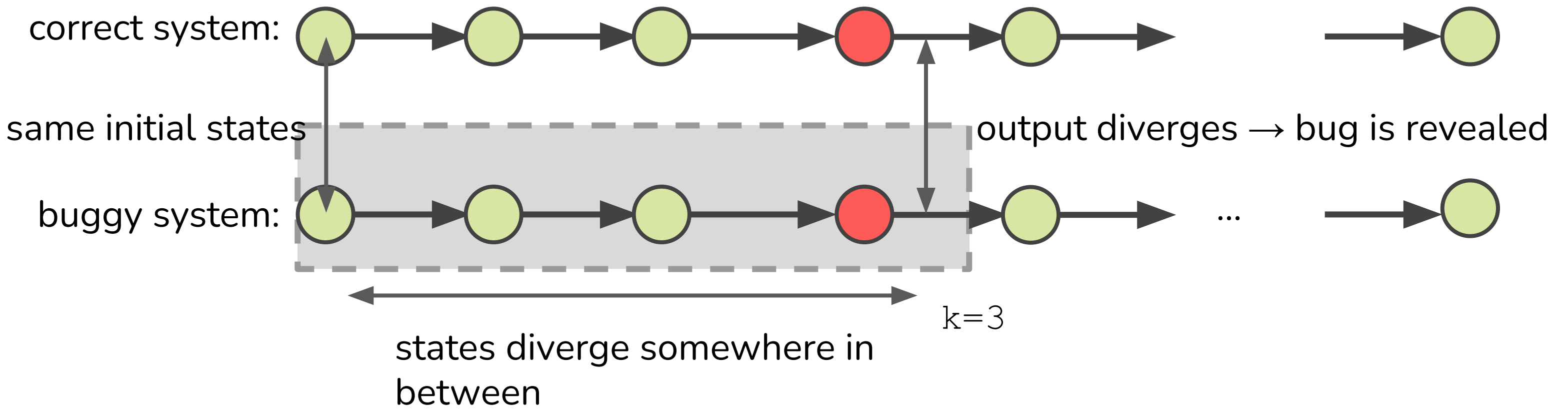
output diverges → bug is revealed

states diverge somewhere in between

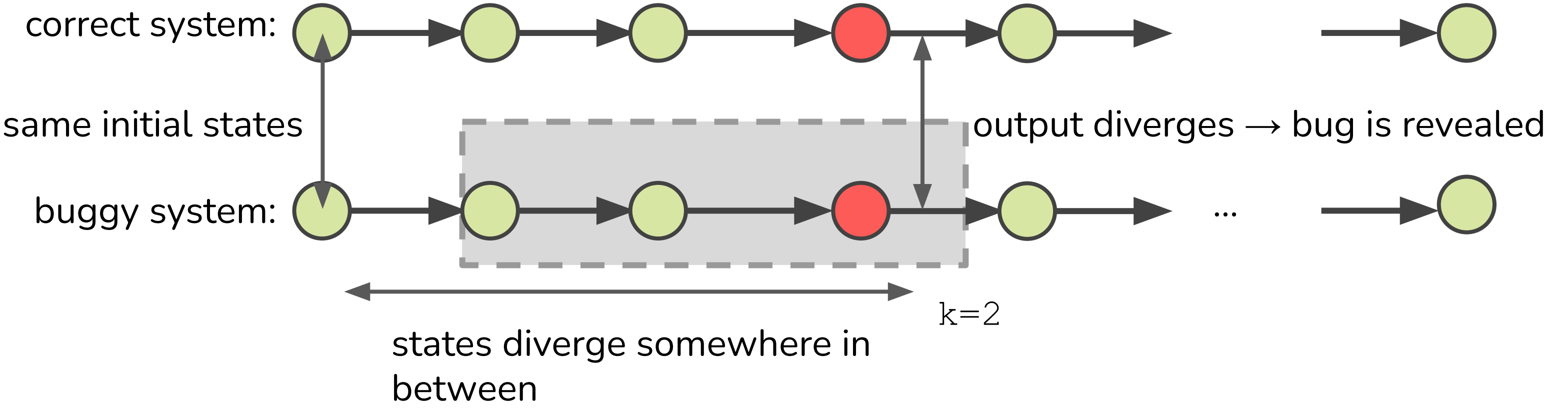
Adaptive Windowing


correct system:
same initial states
buggy system:
output diverges → bug is revealed
...
k=2
states diverge somewhere in between

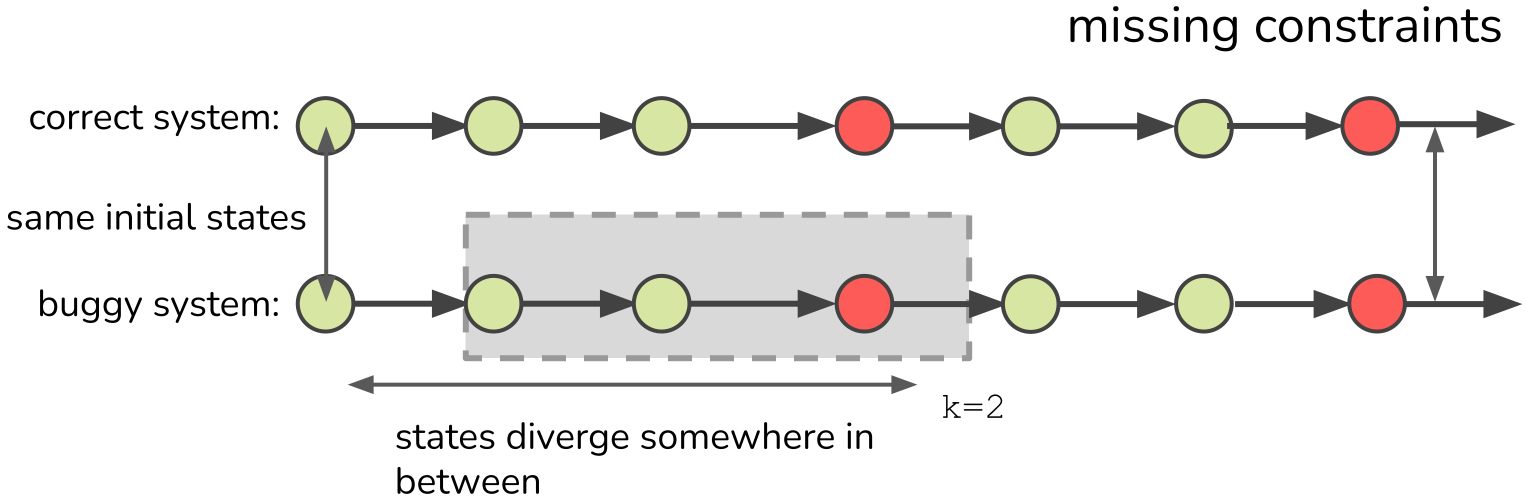
Adaptive Windowing


correct system:
same initial states
output diverges → bug is revealed
buggy system:
...
k=3
states diverge somewhere in between

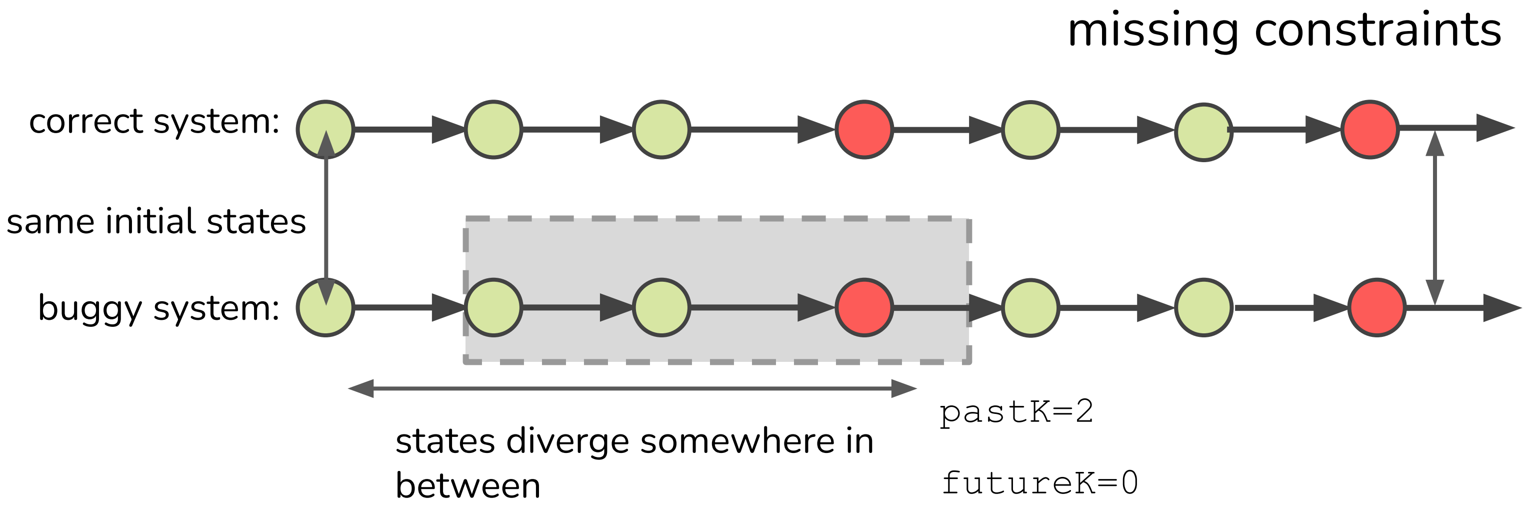
Adaptive Windowing


correct system:
same initial states
buggy system:
output diverges → bug is revealed
...
k=2
states diverge somewhere in between

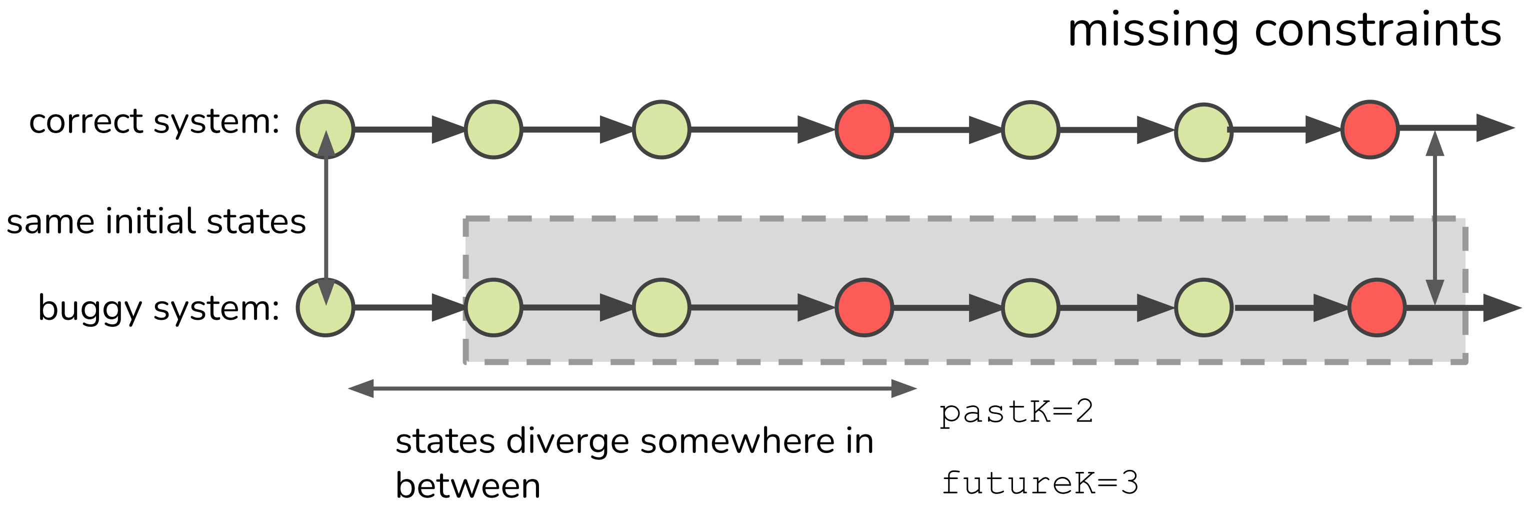
Adaptive Windowing



Adaptive Windowing



Adaptive Windowing



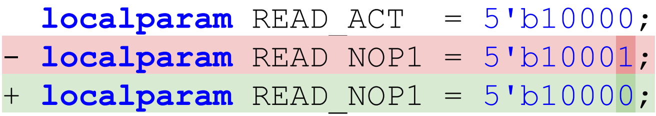
SDRAM Controller Repair

```
  localparam READ_ACT  = 5'b10000;
- localparam READ_NOP1 = 5'b10001;
+ localparam READ_NOP1 = 5'b10000;
```

diff original vs. bug

ASPLOS’22: ❌ Timeout after 12h

SDRAM Controller Repair



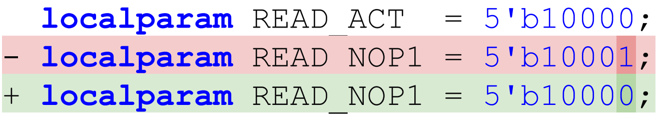
```diff
  localparam READ_ACT  = 5'b10000;
- localparam READ_NOP1 = 5'b10001;
+ localparam READ_NOP1 = 5'b10000;
```

diff original vs. bug

Window Refinement around Step 130

ASPLOS'22: ❌ Timeout after 12h

SDRAM Controller Repair



```
  localparam READ_ACT   = 5'b10000;
- localparam READ_NOP1  = 5'b10001;
+ localparam READ_NOP1  = 5'b10000;
```

diff original vs. bug

Window Refinement around Step 130

```
pastK=0, futureK = 0, k=0
pastK=0, futureK = 2, k=2
pastK=2, futureK = 2, k=4
```

ASPLOS’22: ❌ Timeout after 12h

SDRAM Controller Repair

```
  localparam READ_ACT  = 5'b10000;
- localparam READ_NOP1 = 5'b10001;
+ localparam READ_NOP1 = 5'b10000;
```

diff original vs. bug

```
  READ_ACT: begin
-     next = READ_NOP1;
+     next =  5'b11101;
      // ..
  end
- READ_NOP1: begin
+  5'b11101: begin
      // ..
  end
```

diff bug vs. our repair

Window Refinement around Step 130

```
pastK=0, futureK = 0, k=0
pastK=0, futureK = 2, k=2
pastK=2, futureK = 2, k=4
```

ASPLOS’22: ❌ Timeout after 12h

SDRAM Controller Repair

```
  localparam READ_ACT  = 5'b10000;
- localparam READ_NOP1 = 5'b10001;
+ localparam READ_NOP1 = 5'b10000;
```

diff original vs. bug

```
  READ_ACT: begin
-    next = READ_NOP1;
+    next =  5'b11101;
     // ..
  end
- READ_NOP1: begin
+  5'b11101: begin
     // ..
  end
```

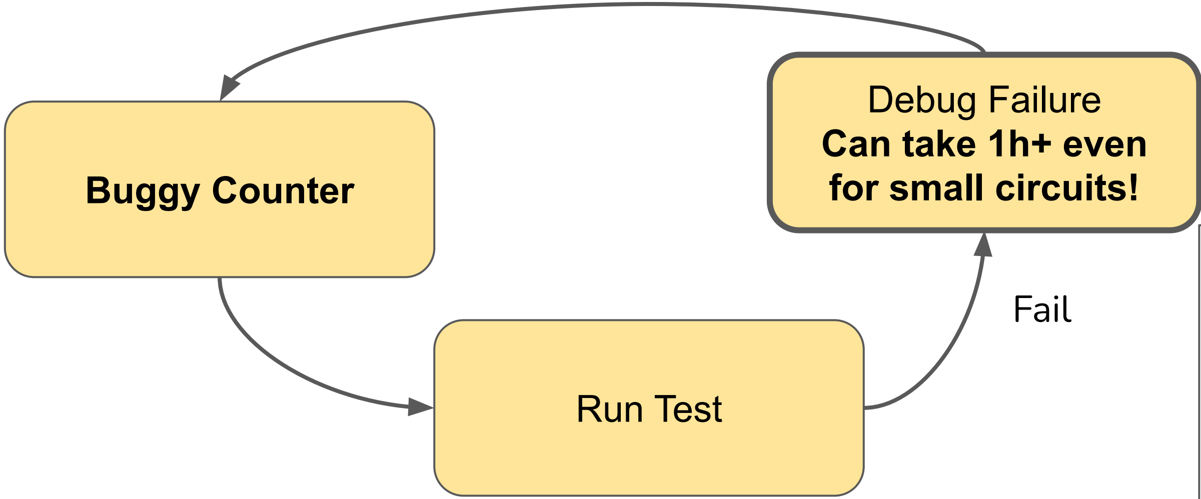
diff bug vs. our repair

Window Refinement around Step 130

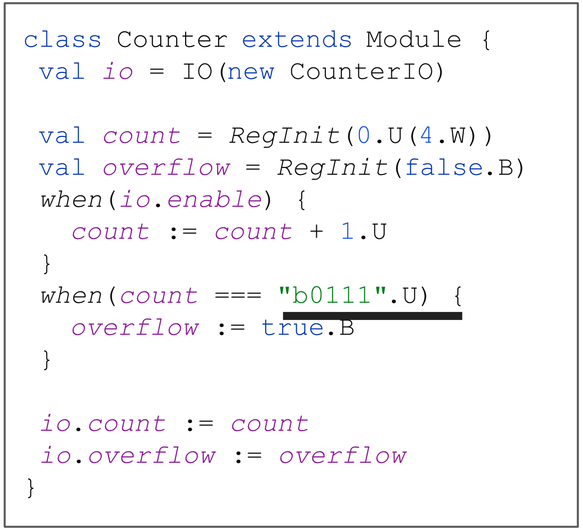
```
pastK=0, futureK = 0, k=0
pastK=0, futureK = 2, k=2
pastK=2, futureK = 2, k=4
```

ASPLOS’22: ❌ Timeout after 12h
Our Tool: ✅ Correct repair in 3s

Automated RTL Repair

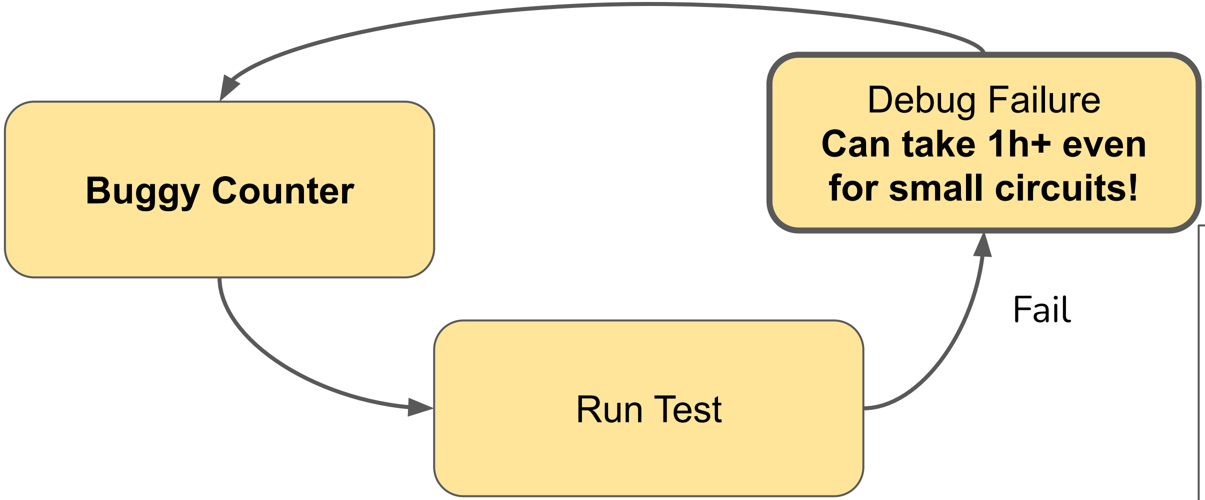


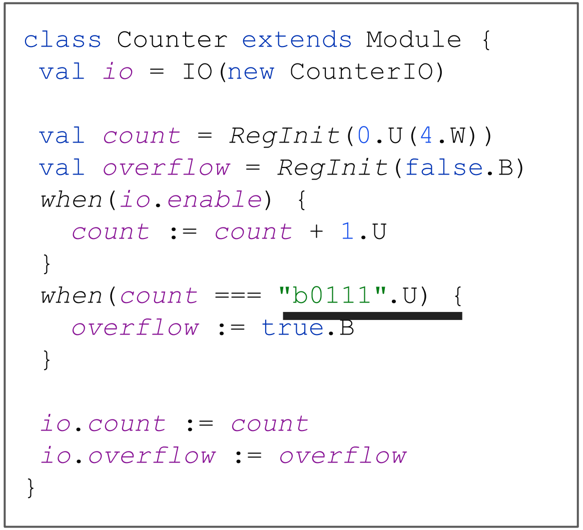
“Have you tried replacing b0111 with b1111”?



```
class Counter extends Module {
 val io = IO(new CounterIO)

 val count = RegInit(0.U(4.W))
 val overflow = RegInit(false.B)
 when(io.enable) {
   count := count + 1.U
 }
 when(count === "b0111".U) {
   overflow := true.B
 }

 io.count := count
 io.overflow := overflow
}
```

Automated RTL Repair



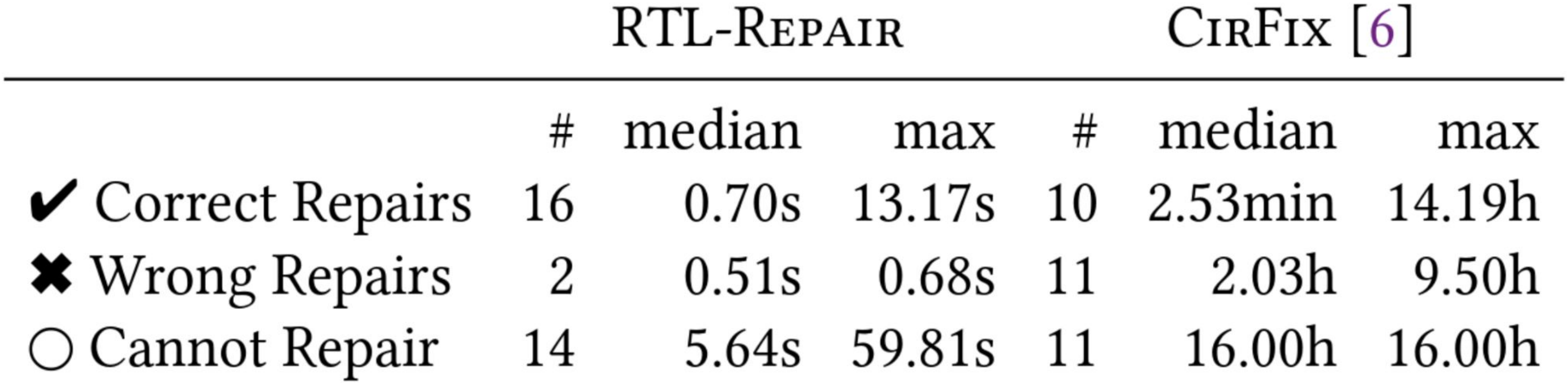


```
class Counter extends Module {
 val io = IO(new CounterIO)

 val count = RegInit(0.U(4.W))
 val overflow = RegInit(false.B)
 when(io.enable) {
   count := count + 1.U
 }
 when(count === "b0111".U) {
   overflow := true.B
 }

 io.count := count
 io.overflow := overflow
}
```

“Have you tried replacing b0111 with b1111”?

Our tool provides this answer in 1s.

Benchmark Overview



	RTL-Repair			CirFix [6]		
	#	median	max	#	median	max
✔ Correct Repairs	16	0.70s	13.17s	10	2.53min	14.19h
✖ Wrong Repairs	2	0.51s	0.68s	11	2.03h	9.50h
○ Cannot Repair	14	5.64s	59.81s	11	16.00h	16.00h

We solve many of the benchmarks, at **interactive speeds**.

RTL-Repair: Fast Symbolic Repair of Hardware Design Code

Kevin Laeufer
laeufer@eecs.berkeley.edu
University of California, Berkeley
Berkeley, CA, USA

Brandon Fajardo*
brfajardo@berkeley.edu
University of California, Berkeley
Berkeley, CA, USA

Abhik Ahuja*
ahujaabhik@berkeley.edu
University of California, Berkeley
Berkeley, CA, USA

Vighnesh Iyer
vighnesh.iyer@eecs.berkeley.edu
University of California, Berkeley
Berkeley, CA, USA

Borivoje Nikolić
bora@eecs.berkeley.edu
University of California, Berkeley
Berkeley, CA, USA

Koushik Sen
ksen@eecs.berkeley.edu
University of California, Berkeley
Berkeley, CA, USA

Featuring numerous repair examples!

Paper PDF

Code on Github

Artifacts Evaluated acm Functional V1.1

Artifacts Available acm V1.1

Results Reproduced acm V1.1

kevinlaeufer.com