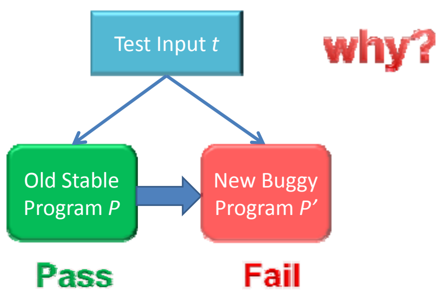
Debugging of Evolving Programs

Abhik Roychoudhury
National University of Singapore
http://www.comp.nus.edu.sg/~abhik

Evolving Programs

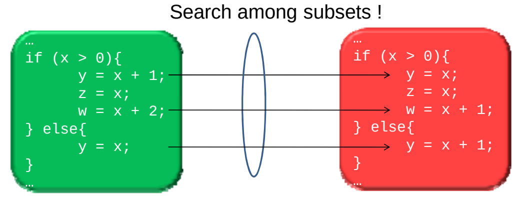
- Code Evolution
  - Consider a banking system
    - Features: Login, Logout, View Balance, …
    - Version 1, P1
  - Customer wants new feature, produce new version P2
    - New feature: Funds transfer
    - This breaks the functionality of "View Balance"
      - No longer see the latest balance correctly!
    - Example of regression due to code evolution
      - Different from "requirements evolution" – intended meaning of "view balance" is unchanged from P1 to P2.

Problem Statement



Change Analysis?

Search among subsets !



Requires defining the set of all changes.

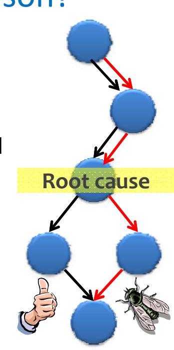
Question: What if the two programs are completely different implementations of the same protocol?

Trace Comparison?

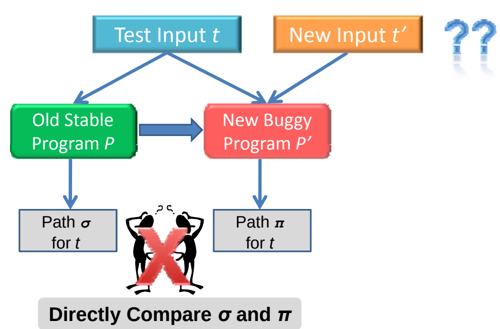
Compare failing test with a **similar, successful** test.

Requirement: How do we find such an execution**?**

Question : why ignore the evolution?

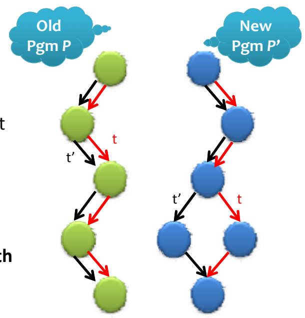


Adapting Trace Comparison

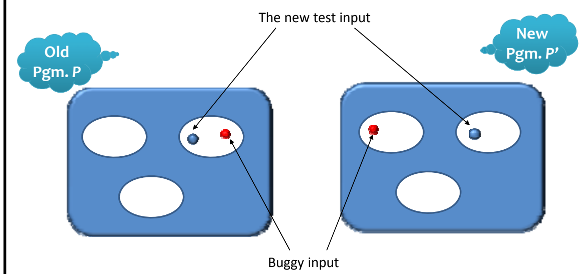


How to obtain the new test?

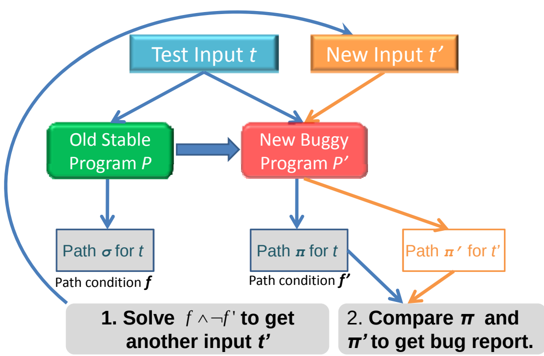
- We have:
  - Two versions of the program. (*P* and *P'*).
  - A test *t* that fails on *P'* but passes on *P*.
- Key requirement: Similarity
  - Test *t* and *t'* are **similar** if they induce
    - **same control flow path in *P*** but
    - **different paths in *P'*.**



How to obtain the new test?



Our Approach



Path conditions

- Quantifier free first order logic formula
- Obtained from a path
  - *input x, y;*
  - *if (x > 1){* $x > 1$
  - *if (y < 20){* $x > 1 \wedge y < 20$
  - ….
- Conjunction of primitive constraints with program variables.
  - All program variables implicitly existentially quantified.

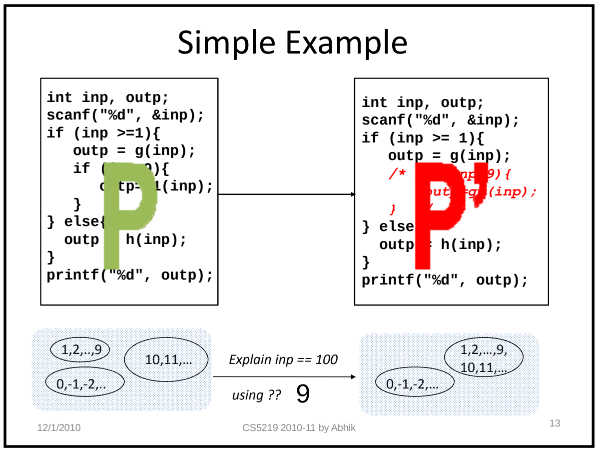
Exercise

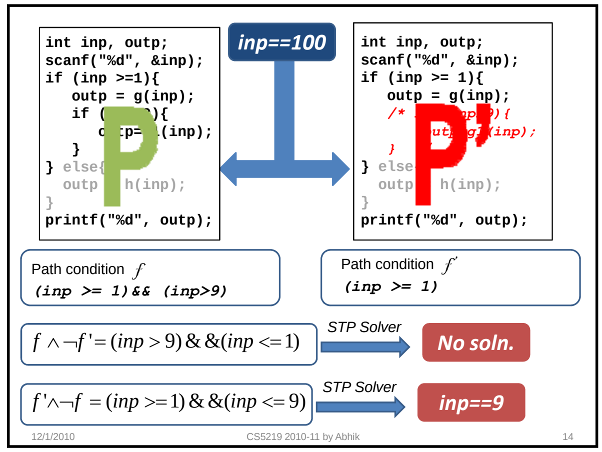
- Given a program and an input, develop an automated method to compute path conditions.
  - *input x, y;*
  - *if (x > 1){* $x > 1$
  - *z = x + y*
  - *if (z < 20){* $x > 1 \wedge x + y < 20$
  - ….

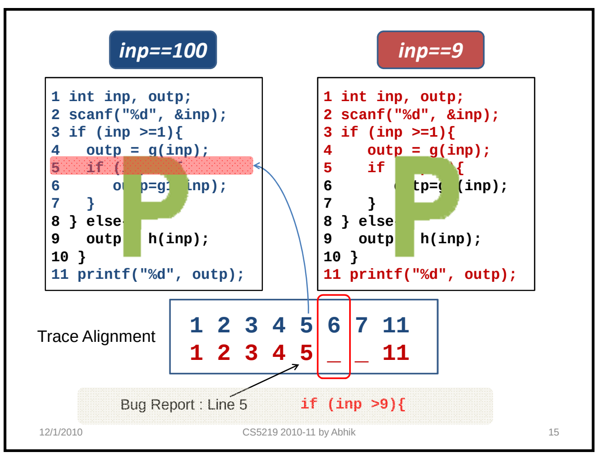
Generating New Input

1. Compute *f*, the path condition of *t* in *P*.
2. Compute *f'*, the path condition of *t* in *P'*.
3. Solve for $f \wedge \neg f'$
   - Many solutions: Compare the trace of each *t'* in *P'* with the trace of *t* in *P'*. Return bug report from *P'*.
   - No solution: go to next step.
4. Solve for $f' \wedge \neg f$
   - Many solutions: Compare the trace of each *t'* in *P* with the trace of *t* in *P*. Return bug report from *P*.
   - No solution: Impossible, since then

$$f \Leftrightarrow f'$$



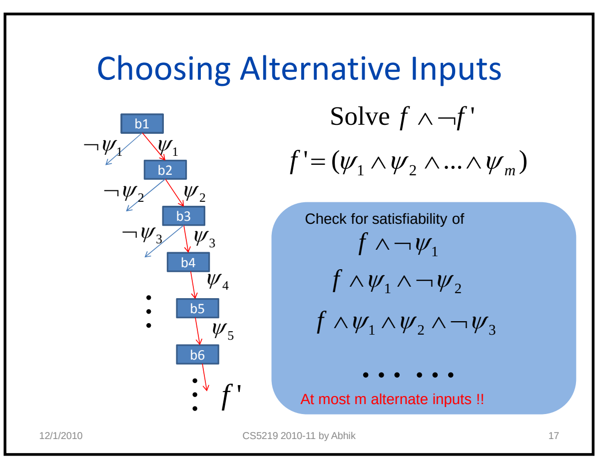


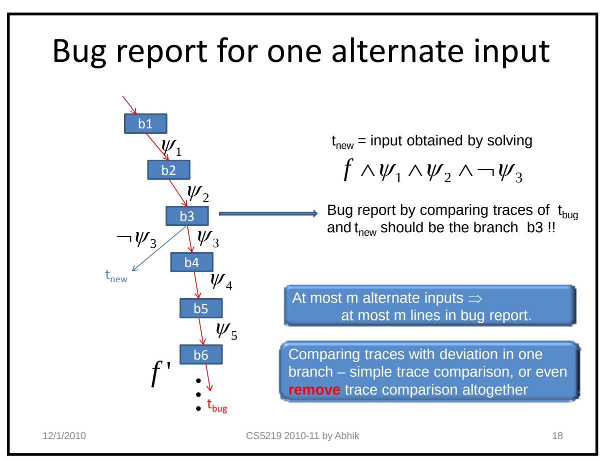


Overview of our Solution

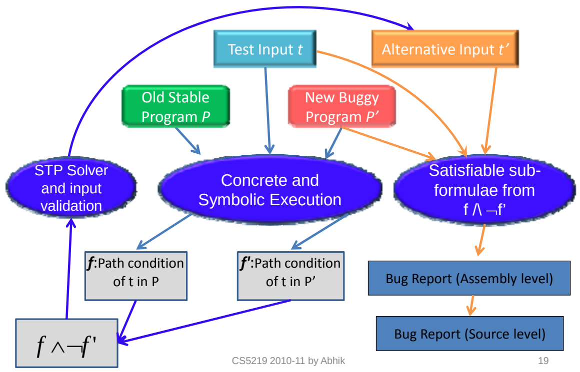
1. Compute f, the path condition of t in P.
2. Compute f', the path condition of t in P'.
3. Solve for $f \wedge \neg f'$.
   - Many solutions: Compare the trace of each t' in P' with the trace of t in P'. Return bug report from P'.
   - No solution: go to next step.
4. Solve for $f' \wedge \neg f$.
   - Many solutions: Compare the trace of each t' in P with the trace of t in P. Return bug report from P.
   - No solution: Impossible, since then $(f \Leftrightarrow f')$

12/1/2010 CS5219 2010-11 by Abhik 16





Putting Everything Together

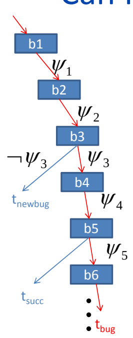


Results

Buggy Program	Stable program	Time taken	Bug report size
LibPNG v1.0.7 (31164 loc)	LibPNG v1.2.21 (36776 loc)	13 m 34 s	9
TCPflow (patched)	TCPflow (unpatched)	31m	6
Miniweb (2838 loc)	Apache (358379 loc)	14s	5
Savant (8730 loc)	Apache httpd (358379 loc)	9m	46

If we require the alternative input to behave the same in buggy program and reference program (passing test) -
the **bug report size is 1 in all three cases.**

Can find other potential bugs




```
LibPNG v1.0.7  length =
        png_get_uint_32(chunk_length);
LibPNG v1.2.21 length =
        png_get_uint_31(chunk_length);

png_get_uint_31(png_structp png_ptr,
                png_bytep buf)
{
  png_uint_32 i = png_get_uint_32(buf);
  if (i > PNG_UINT_31_MAX)
     png_error(png_ptr, "PNG unsigned integer
                        out of range.");
  return (i);
}
```


Summary so far

- Novel approach for debugging evolving programs
  - Semantic analysis to generate alternative similar inputs
  - Can be applied to two totally different implementations.
- Implementation and Evaluation
  - DARWIN tool : built on BitBlaze platform.
  - Accurate bug reports for various real-life examples.
- Extensions
  - Found new bugs via debugging of a given observable error
  - More detailed path conditions via pred. instrumentation.

An experiment

Validate Embedded Linux

AGAINST

Linux (GNU Core-utils, net –tools)

Busybox distribution is 121 KLOC.
Various errors to be root-caused in `tr`, `arp`, `top`, `printf`.

Trying on Embedded Linux

- The concept
  - Golden: GNU Coreutils, net-tools
  - Buggy: Busybox
    - De-facto distribution for embedded devices.
    - Aims for low code size
    - Less checks and more errors.
  - Try DARWIN!
- The practice
  - Failing input takes logically equivalent paths in Busybox and Core-utils.

Going beyond

P
```
input x;
y = 2 * x;
output y
```

P'
```
input x;
y = 2*x + 1;   // bug
output y
```

Observable error: Input x == 0, Expected output y == 0
Observed output y == 1

Employ DARWIN:
In program P, path condition f == true
path condition f' == true
f ∧ ¬ f' == false also f' ∧ ¬f == false.
No Bug report generated !!

What went wrong?

- The effect of the bug is not observable in terms of change in program paths.

P
```
input x;
y = 2 * x;
output y
```

P'
```
input x;
y = 2*x+1;   // bug
output y
```

- Reasoning with path conditions does not expose the bug location either!

A more direct approach

P
```
input x;
y = 2 * x;
output y
```

P'
```
input x;
y = 2*x+1;   // bug
output y
```

- Characterize observable error (obs)
  - y != 0
- Weakest pre-condition along failing path w.r.t. obs
  - 2*x != 0
  - 2*x + 1 != 0
- Compare the WPs and find differing constraints.
- Map differing constraints to the lines contributing them.

Weakest pre-condition

Along a program path.

input x, y;	
x = x +1;	z – 1 < 0 ∧ x + 1 + y > 0
if (x + y > 0){	z – 1 < 0 ∧ x + y > 0
z = z – 1;	z- 1 < 0
}	
else {… }	
print z;	z < 0
	z < 0

WP along a path

- Along a path
  - Start with a primitive constraint
    - You are seeking to explain under what situations it will hold at the end of the program path.
  - Proceed along the path from the end.
    - For every branch, conjoin the branch condition.
    - For every assignment, replace occurrences of the lhs by rhs in the existing formula.
  - Stop when you reach the beginning of the path.
    - What kind of a constraint will you end up with?

Entire failing trace is not needed

```
1. … // input inp1, inp2
2. if (inp1 > 0)
3.    x = f1(inp1);   // bug
4. else x = g1(inp1);
5. if (inp2 > 0)
6.    y = f2(inp2)
7. else y = g2(inp2);
8. … // output x, y    observe unexpected x < 0 for inp1 == inp2 == 1
```

Observable error: x<0 at line 8.
WP along the trace of inp1 == inp2 == 1 gives us
inp2 > 0 ∧ inp1 > 0 ∧ f1(inp1) < 0
Points to lines { 2, 3, 5 }
Line 5 is clearly not relevant since inp2 does not contribute to computing x.

What is the issue?

- Inherent parallelism exists in sequential programs
  - inp1 helps compute x
  - inp2 helps compute y
- Exploit the inherent parallelism to project the "relevant" part of the trace.
  - **Dynamic slicing (from last lecture!)**
  - Symbolic execution (WP computation) along the dynamic slice only.
- Crucial for scalability of our method!

Approach 2 – in action (simplified)

```
1. ... // input inp1, inp2           f1(inp1)< 0 ∧ inp1> 0
2. if (inp1 > 0)                     f1(inp1) < 0 ∧ inp1 > 0 (control dep.)
3. x = f1(inp1);   // bug            f1(inp1) < 0  (data dep.)
4. else x = g1(inp1);
5. if (inp2 > 0)
6. y = f2(inp2)
7. else y = g2(inp2);
8. ... // output x, y    observe unexpected x < 0 for inp1 == inp2 == 1
```


Approach 2 - summary

- Set observable error: x< 0
- Set slicing criterion: value of x at line 8
- Simultaneously perform
  - Dynamic slicing – Control and Data dependencies
  - Symbolic execution – along the slice
    - WP computation along the slice
- The above is performed on both P, P'
  - Produces WP, WP' – conjunction of constraints
  - Find differing constraints in WP, WP'
  - Map differing constraints to contributing LOC – this is the bug-report.

Comparing WP, WP'

- WP = $(\varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_n)$
- WP' = $(\varphi'_1 \wedge \varphi'_2 \wedge \ldots \wedge \varphi'_m)$
- Check
  - WP $\Rightarrow \varphi'_1$ ...
  - WP' $\Rightarrow \varphi_1$ ...
  - Solver may choke!
- Instead, we can perform pair-wise comparison
  - Too costly ??

Comparing WP, WP'

- WP = $(\varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_n)$
- WP' = $(\varphi'_1 \wedge \varphi'_2 \wedge \ldots \wedge \varphi'_m)$
- Pair-wise comparison of constraints can blow-up.
- Tautology elimination – more than 90% reduction!

```
X = 1
...
if (X > 0) {
  ...
  printf("%d", Y);
}
```

WP computation along slice:

1 > 0 ∧ Y < 0 // due to assignment of X

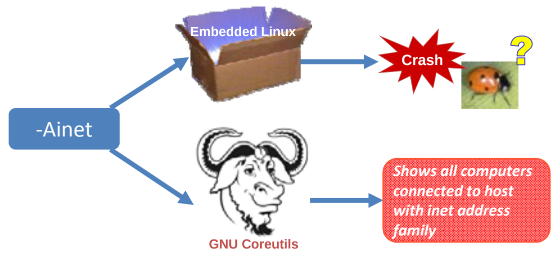
X > 0 ∧ Y < 0 // due to the branch

Y < 0 // the constraint we start with

So, what do we do then?

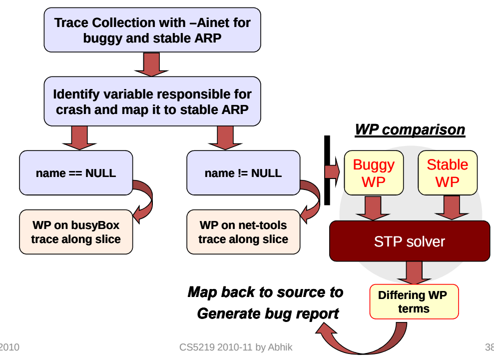
- WP = Conjunction of n constraints
  - Remove tautologies
  - WP = $(\varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_x)$ $x < n$
- WP' =conjunction of m constraints
  - Remove tautologies
  - WP = $(\varphi'_1 \wedge \varphi'_2 \wedge \ldots \wedge \varphi'_y)$ $y < m$
- For each φ'_i check if there is a φ_j s.t. $\varphi_j \Rightarrow \varphi'_i$
- For each φ_i check if there is a φ'_j s.t. $\varphi'_j \Rightarrow \varphi_i$

A glimpse inside the ARP bug



Crash identified as NULL pointer access at crash site
hw_type unexpectedly set as NULL at crash site

Approach 2 in action



Summarizing

- Debugging evolving programs (code evolution)
  - Program Versions
  - Embedded SW against non-embedded version
  - Two implementations of same specification
    - Web-servers implementing http protocol
- Use of formal techniques into debugging
  - Beyond a "black art".

For more …

[FSE09] DARWIN: An Approach for Debugging Evolving Programs Dawei Qi, Abhik Roychoudhury, Zhenkai Liang, Kapil Vaswani, ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE), ESEC-FSE 2009.
- http://www.comp.nus.edu.sg/~abhik/pdf/fse09.pdf

Also see:
Yesterday my program worked. Today it does not. Why?
Andreas Zeller, [ESEC-FSE 1999].
- http://www.infosun.fim.uni-passau.de/st/papers/tr-99-01/esec99-talk.pdf

[FSE10] Golden Implementation Driven Software Debugging Ansuman Banerjee, Abhik Roychoudhury, Johannes A. Harlie, Zhenkai Liang, ACM SIGSOFT Symposium on Foundations of Software Engineering (FSE) 2010.
- http://www.comp.nus.edu.sg/~abhik/pdf/fse10.pdf