

# Automatic Rootcausing for Program Equivalence Failures in Binaries

**Shuvendu K. Lahiri** (Microsoft Research)

Rohit Sinha (UC Berkeley)

Chris Hawblitzel (Microsoft Research)

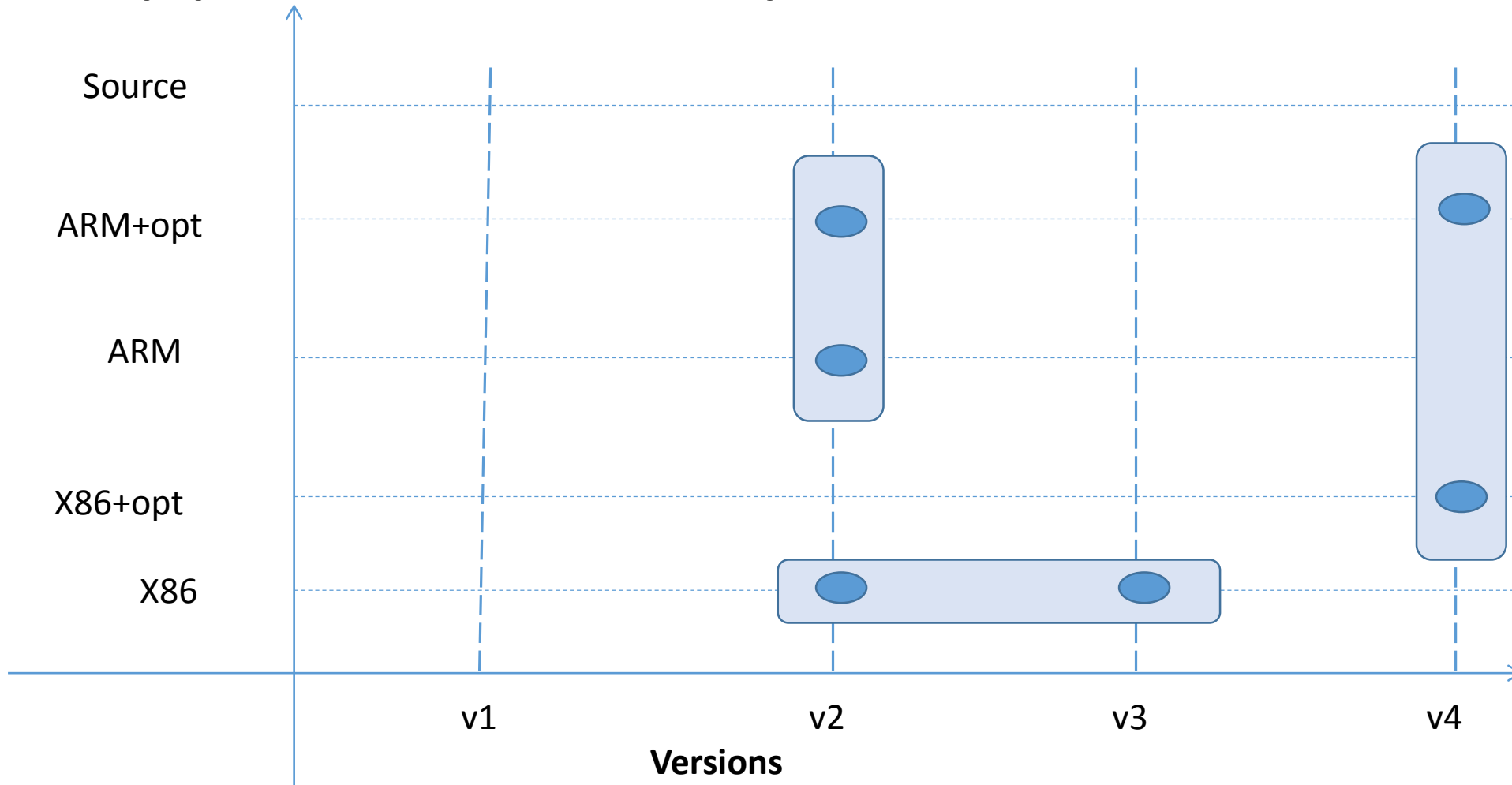
# Large scale program equivalence checking

- Compiler translation validation
  - [Pnueli et al. TACAS'98, Necula PLDI'00, ...]
- Cross-version verification
  - [Godlin & Strichman DAC'09, Lahiri et al. CAV'12,...]
- Verifying student solutions against reference implementations
  - [Singh et al. PLDI'13]

# Motivation: Rootcausing equivalence failures

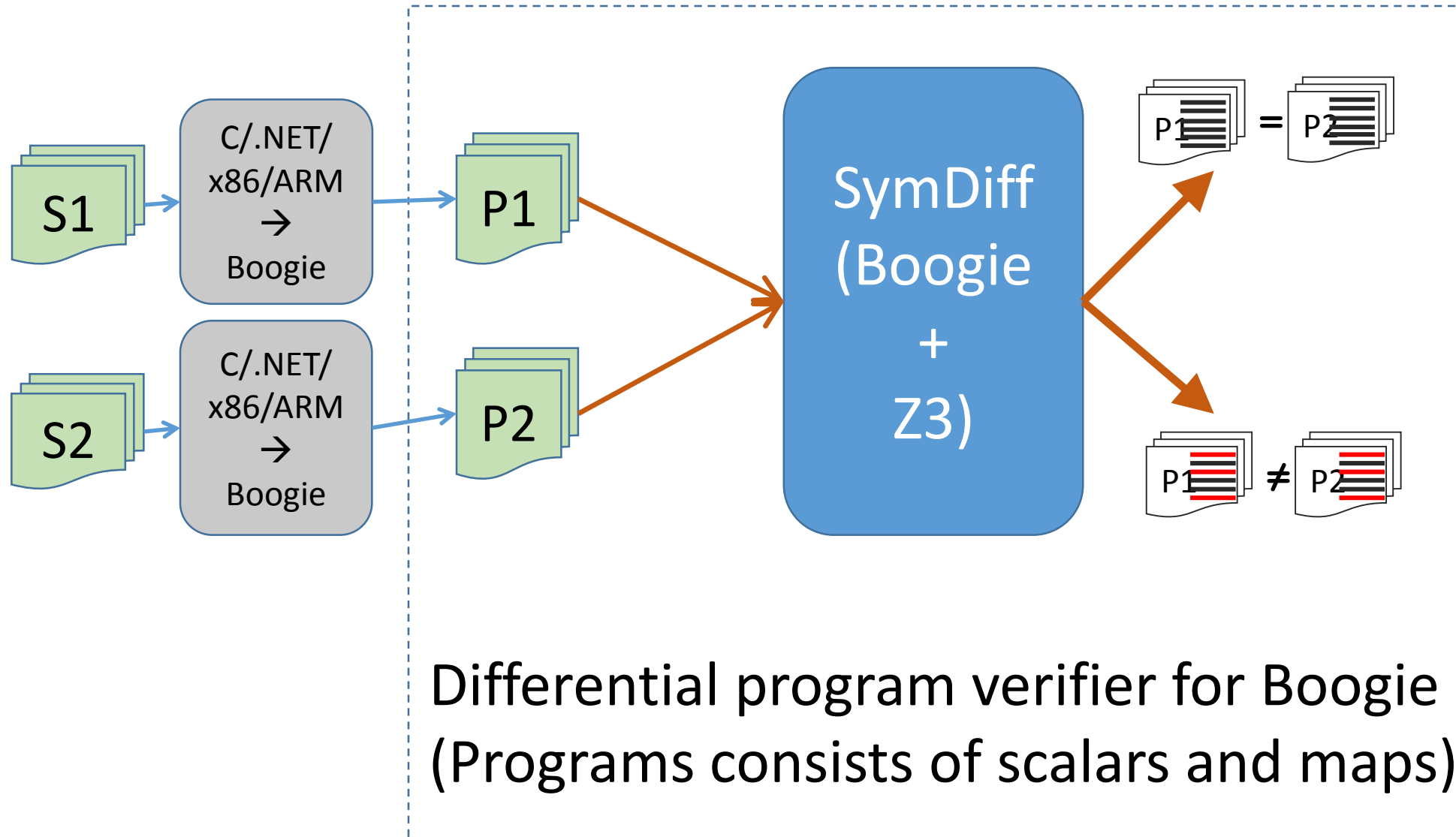
- Provide effective feedback to users of the tool
  - Dealing with **thousands of equivalence failures**
- Compiler translation validation
  - Same alarm manifests in hundreds of test programs
- Comparing student attempts
  - Many students often make similar mistakes

# Application: Compiler validation



- Validating the CLR .NET compiler [Hawblitzel, Lahiri et al. FSE'13]
  - SymDiff to compare two assembly/binary programs

# Verification flow



# Bucketization of equivalence failures

- Almost 500,000 test C# methods pushed through the tool
  - Was applied by CLR test team for several months, found real bugs
  - Even 1-2% false alarm → ~several thousand warnings
- **Main ask from users**
  - Need to group failures into a small number of buckets
- Each bucket captures one source of equivalence failure
  - Different manifestations of the same bug
  - Different manifestations of same false alarm

# False alarms

- Often due to **modular checking** and **missing domain-knowledge**
  - Concrete addresses
    - 0x004fe208 vs 0x003dd484
  - Different memory layout by the compiler
    - Field stored in two different offsets: [eax + 4] vs. [eax + 32]
  - Aliasing assumptions known only to the compiler
    - Store to address x does not modify address y
  - Side effects of procedures
    - A procedure call does not modify certain heap locations
    - Purity
  - ....

# BUCKETS

File Edit View Favorites Tools Help

Suggested Sites Web Slice Gallery

**BUCKETS: All (177)**

**CHILDREN (165)**

- [Direct vs. indirect call \(OffsetOf\) \(86\)](#)
- [Different offsets \(12\)](#)
- [MulticastDelegate vs. constructor \(11\)](#)
- [Mismatched call to CORINFO\\_HELP\\_CHKCASTCLASS \(10\)](#)
- [Unknown \(10\)](#)
- [Switch table \(10\)](#)
- [Different constants \(6\)](#)

# Syntax diffs

File Edit View Favorites Tools Help

Suggested Sites Web Slice Gallery

**Counterexamples + syntactic rootcause**

```
47: 000038 2D00  cmp  x5, 0
48: 00003A D005  beq  SHORT G_M55613_IG07
49: 00003C 4E28  mov  r0, r5
50: 00003E 2100  movs r1, 0
51: 000040 6803  ldr  x3, [r0]
52: 000042 6B1B  ldr  x3, [r3+48]
53: 000044 68DB  ldr  x3, [r3+12]
54: 000046 4798  blx  x3 // ck: bj(bool)
55:
56: G_M55613_IG07:
57: 000048 6D23  ldr  r3, [r4+80]
58: 00004A 3301  adda r3, 1
59: 00004C 6523  str  r3, [r4+80]
60: 00004E 4620  mov  r0, r4
61: 000050 F000 F800 b1  bk: au() : ref
[calllee = 2065, r0_1416927 = -844, Mem = 1329, flags = 0] 1336, flags = 0]
Mismatch (final memory state vs. final memory state)
Cause: Different Functions (CallMem_const_bk$003A$ref vs CallMem_const_CORINFO_HELP_CHKCASTCLASS)
Instruction 1 :: 61: 000050 F000 F800 b1  bk: au() : ref
Instruction 2 :: 17: ( ) b1 (Symbol CORINFO_HELP_CHKCASTCLASS)
62: 000054 6803  ldr  r3, [r0]
63: 000056 F000 F800 b1  System.Collections.Generic.
List<System. Canon: get_Count(): int
64: 00005A 6D23  ldr  r3, [r4+80]
65: 00005C 4298  cmp  r0, r3
66: 00005E D80C  hls  SHORT G_M55613_IG08
67: 000060 4620  mov  r0, r4
68: 000062 F000 F800 b1  bk: au() : ref
69: 000066 6D21  ldr  r1, [r4+80]
70: 000068 6803  ldr  r3, [r0]
71: 00006A F000 F800 b1  System.Collections.Generic.
List<System. Canon: get_Item(int): ref
72: 00006E 4601  mov  r1, r0
73: 000070 F104 0040 add  r0, r4, 64
74: 000074 F000 F800 b1  CORINFO_HELP_ASSIGN_REF
75: 000078 E001  b  SHORT G_M55613_IG09
76:
77: G_M55613_IG08:
78: 00007A 2300  movs r3, 0
79: 00007C 6423  str  r3, [r4+64]
80:
81: G_M55613_IG09:
82: 00007E 6C21  ldr  r1, [r4+64]
83: 000080 4620  mov  r0, r4
84: 000082 F000 F800 b1  c: e(ref): ref
85: 000086 4601  mov  r1, r0
86: 000088 2900  cmp  r1, 0
87: 00008A D80C  beq  SHORT G_M55613_IG13
88:
89: G_M55613_IG10:
90: 00008C 680B  ldr  x3, [x1]
91: 00008E F240 0200 movw r2, LOW_RELOC 0x20B786C
92: 000090 F2C0 0200 movw r2, HIGH_RELOC 0x20B786C
93: 000096 4293  cmp  x3, r2
94: 000098 D100  bne  SHORT G_M55613_IG12
95:
96: G_M55613_IG11:
97: 00009A E006  b  SHORT G_M55613_IG13
18: ( ) mov r5 r0
19: (a) cmp r5 0
20: (eq) b (Symbol L0)
21: ( ) mov r0 r5
22: ((s) mov r1 0)
23: ( ) ldr r12 (Ptr r0)
24: ( ) ldr r12 (Ptr r12 48)
25: ( ) ldr r12 (Ptr r12 (OffsetOf "ck: bj(bool)"))
26: ( ) blx (Typed "ck: bj(bool)" r12)
27: (Line 29)
28: (Label L0)
29: ( ) ldr r3 (Ptr r4 80)
30: ((a) add r3 r1)
31: ( ) str r3 (Ptr r4 80)
32: ( ) mov r0 r4
33: ( ) b1 (Symbol "bk: au() : ref")
34: ( ) ldr r12 (Ptr r0)
35: ( ) ldr r12 (Ptr r12 44)
36: ( ) ldr r12 (Ptr r12 (OffsetOf "System.Collections.Generic.List<System. Canon: get_Count(): int"))
37: ( ) blx (Typed "System.Collections.Generic.List<System. Canon: get_Count(): int" r12)
38: ( ) ldr r3 (Ptr r4 80)
39: ((a) cmp r0 r3)
40: ((le) b (Symbol L1))
41: ( ) mov r0 r4
42: ( ) b1 (Symbol "bk: au() : ref")
43: ( ) ldr r12 (Ptr r4 80)
44: ( ) ldr r12 (Ptr r0)
45: ( ) ldr r12 (Ptr r12 48)
46: ( ) ldr r12 (Ptr r12 (OffsetOf "System.Collections.Generic.List<System. Canon: get_Item(int): ref"))
47: ( ) blx (Typed "System.Collections.Generic.List<System. Canon: get_Item(int): ref" r12)
48: ( ) mov r1 r0
49: ( ) add r0 r4 64
50: ( ) b1 (Symbol CORINFO_HELP_ASSIGN_REF)
51: ( ) b (Symbol L2)
```



# Prior works

**This work**  
Verified rootcause  
(for equivalence)

**Fault localization**  
(Jose et al. PLDI'11, ..)

**Program Repair**  
(Nguyen et al. ICSE'13, Singh et al. PLDI'13, ...)

- Provides a program slice (often large for eq checking of binaries)
- Little guarantee

- Need to **repair** the program
- Needs a template of repair
- Scalability

# Wish list for rootcausing

- **Formalize** a valid rootcause at Boogie level (and thus can verify)
  - Points out the **first pair of instructions** where programs diverge
- **Automatic**
  - Providing templates difficult for failures due to modeling imprecision
- Can express **domain knowledge at the Boogie level**
  - Ideas can be agnostic to the source programs
  - Reusable for other programs (x86/ARM/x64/C/Java)

# This talk

- Formalization of rootcause for equivalence failures
  - For structurally similar programs
- Dealing with the need for multiple fixes
- Implementation
  - Optimizations (MAXSAT, Binary search)
- Evaluation

# Example

Equivalence does not hold if f can modify M1 at x

```
procedure p1(M:[int]int, x:int)
returns (r1:int, M1:[int]int)
{
  M1 := M;
  r5 := M1[x]; //ld r5, [x]

  M1, r1 := f(M1, r5); //call f
  r1 := r5; //mov r1, r5
  M1, r1 := f(M1, r1); //call g
  return;
}
```

```
procedure p2(M:[int]int, x:int)
returns (r2:int, M2:[int]int)
{
  M2 := M;
  r6 := M2[x]; //ld r6, [x]

  M2, r2 := f(M2, r6); //call f
  r2 := M2[x]; //ld r2, [x]
  M2, r2 := g(M2, r2); //call g
  return;
}
```

**Optimized**



Insight: Fix p2 by using values computed by p1

**Optimization:** Remove redundant load

# Rootcause definition

- For procedures P1, P2 and a counterexample, a *fix*  $(L,R)$  is a pair of assignments  $L: x := e$  (in P1), and  $R: y := e'$  (in P2) such that
  1. replacing  $e'$  (in P2) with *the value of*  $e$  at L (in P1), makes P1 and P2 equivalent, and
  2.  $(L, R)$  is the *earliest* such pair satisfying (1)
- Note that a “fix” does not repair P2
  - Not the same as replacing expression  $e'$  with  $e$

# Example

Captures the value from p1

Replaces the value in p2

```
const r5@0: int;
procedure p1(M:[int]int, x:int)
returns (r1:int, M1:[int]int)
{
  M1 := M;
  r5 := M1[x];           //ld r5, [x]
  assume r5@0 == r5;

  M1, r1 := f(M1, r5); //call f
  r1 := r5;           // mov r1, r5
  M1, r1 := f(M1, r1); //call g
  return;
}
```

**Optimized**



```
procedure p2(M:[int]int, x:int)
returns (r2:int, M2:[int]int)
{
  M2 := M;
  r6 := M2[x];           //ld r6, [x]

  M2, r2 := f(M2, r6); //call f
  r2 := M2[x];           //ld r2, [x]
  r2 := r5@0;

  M2, r2 := g(M2, r2); //call g
  return;
}
```

**Extra load**

**Optimization:** Remove redundant load

# Instrumentation

- Left program

For each scalar assignment instruction  $l : \mathbf{x} := \mathbf{e}$  with a label  $l \in L$ , we transform it to:

$$l : \mathbf{x} := \mathbf{e}; \text{assume}(\theta@l = \mathbf{x})$$

- Right program

For each assignment instruction  $r : \mathbf{x} := \mathbf{e}$  with a label  $r \in R$ , we transform it to:

$$r : \mathbf{x} := \mathbf{e}; \mathbf{x} := \gamma_r? \theta@r : \mathbf{x}; \text{assume} \bigwedge_{l \in L} (\beta_r^l \Rightarrow \mathbf{x} = \theta@l)$$

# Benefit of the formulation

- Naturally captures debugging equivalence failures
  - Provides a program pair that helps with debugging
  - Useful for bucketization
- Automatic (when such a pair exists)
- Do not need to solve the (more difficult) *repair* problem
- Exploits the *similarity of computations* on both sides



# Challenge: multiple fixes

- Two cases
  - Single fix along multiple paths
  - Multiple fixes along single path

# Single fix along multiple paths

- Fix *only one path* in the **left** program
  - Formally: rootcause verified for all inputs that exercise the counterexample path in the left program (weaker guarantee than all inputs)
  - May take the **right** program along a different control flow path
- Exploits the structural similarity of the two programs
  - Unlike previous work that treats one program as a black-box [Singh et al. PLDI'13], hence need to **repair the entire program**

# Multiple fixes along a path

- Encode domain knowledge as additional preprocessing
  - Weaken the equivalence check
1. Fix *intermediate synchronization* points
    - E.g. State of the heap has to be identical after procedure calls
    - Weaken the final equivalence check with intermediate equivalence that failed
  2. Constrain callee summaries

Can be expressed  
as preprocessing  
of Boogie  
programs

# Constraining Callees : Weaker Fix

```
procedure p1(M:[int]int, x:int)
returns (r1:int, M1:[int]int)
{
  M1 := M; r1 := M1[x];
  assume M1@0 == M1;

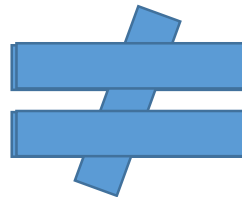
  M1, r1 := getLength(M1, r1);

  assume M1 == M1@0; //no side-effect on heap
  assume r1@0 == r1;

  if (r1 > 0) {
    M1, r1 := writeToFile(M1, r1);
  }
  ...
}
```

```
procedure p2(M:[int]int, x:int)
returns (r2:int, M2:[int]int)
{
  M2 := M; r2 := M2[x];
  r2 := M2[r2 + 8];
  r2 := r1@0;

  if (r2 > 0) {
    M2, r2 := writeToFile(M2, r2);
  }
  ...
}
```



**Domain knowledge:** If a callee (e.g. `getLength`) appears in only one of the programs, treat it as side-effect-free

# MAXSAT-based optimization

- If P1 and P2 have  $n$  assignments each, our naïve algorithm explores  $O(n^2)$  candidate fixes.
- Only a small set of candidate pairs actually fixes P2
  - How do we prune away the rest? (difficult to get concrete runs due to uninterpreted functions)
- Pose it as a MAXSAT problem for any assignment in P2

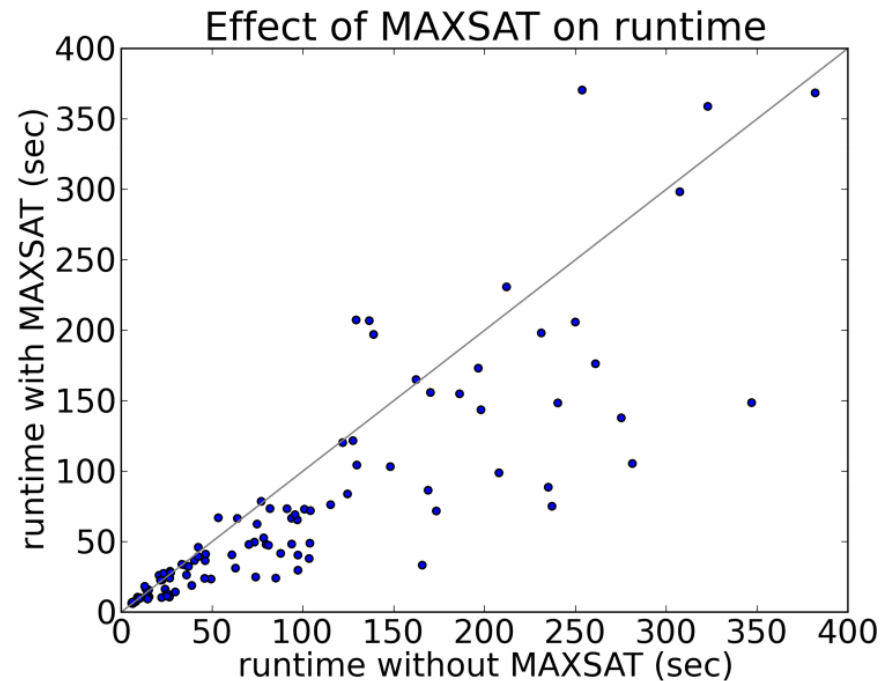
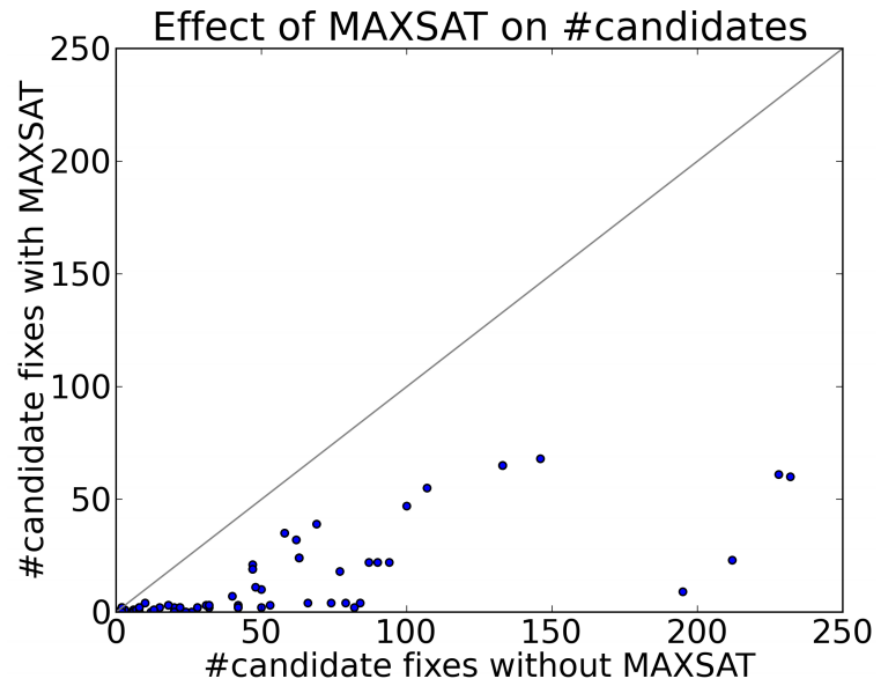
For each assignment instruction  $r : x := e$  with a label  $r \in R$ , we transform it to:

$$r : x := e; x := \gamma_r? \theta@r : x; \text{assume } \bigwedge_{l \in L} (\beta_r^l \Rightarrow x = \theta@l)$$

What is the maximal subset of conjuncts that satisfies P1 != P2

# Effect of MAXSAT optimization

- Average 49% improvement in runtime, and 4x reduction in the number of candidates



# Evaluation

- A representative sample of benchmarks from earlier work
- **JIT vs. compiled binaries**
  - Average of 165 assembly instructions (1242 Boogie statements) per procedure
  - Found rootcause in 34/46 benchmarks (74% of cases)
- **x86 vs. Optimized x86**
  - Average of 68 assembly instructions (510 Boogie statements) per procedure
  - Found rootcause in 12/15 small benchmarks (80% of cases)

# Conclusion

- Natural formulation of verified rootcause for equivalence failures of similar programs
  - Automatic
  - Can be extended to several cases requiring multiple fixes
  - Rootcause integrated into SymDiff Codeplex
- Future Directions:
  - Combine with CEGIS (multiple fixes)
  - Application to automatic grading of student submissions in MOOCs

<http://research.microsoft.com/symdiff>



# Questions