# Differential program verification

**Shuvendu K. Lahiri**

Research in Software Engineering (RiSE),

Microsoft Research

Redmond, WA

# Involved in building static assertion checkers

**HAVOC [POPL'06,'08,'09, CAV'09, VSTTE'10, S&P'13]**

❑ *Heap logics, efficient memory modeling for C*

- NTFS object management(~50 bugs, 300KLOC)
- Variants of MSRC Security Vulnerabilities in IE/Kernel (~100 bugs, ~2MLOC)

Uses components Z3, Boogie, Corral, Houdini, …

**STORM [CAV'09]**

❑ *Reducing concurrency analysis to sequential*

- Concurrency bugs in Drivers (~10 bugs, 10KLOC)

**Angelic Verifier [PLDI'13,CAV'15]**

❑ *Configurable angelic environment specification inference*

- Assertion checker (memory safety, type-state) on Drivers/Kernel (~100+ bugs, ~100KLOC) with minimal env modeling
- Will ships in a future release of Windows DK

# Challenges for static assertion checkers

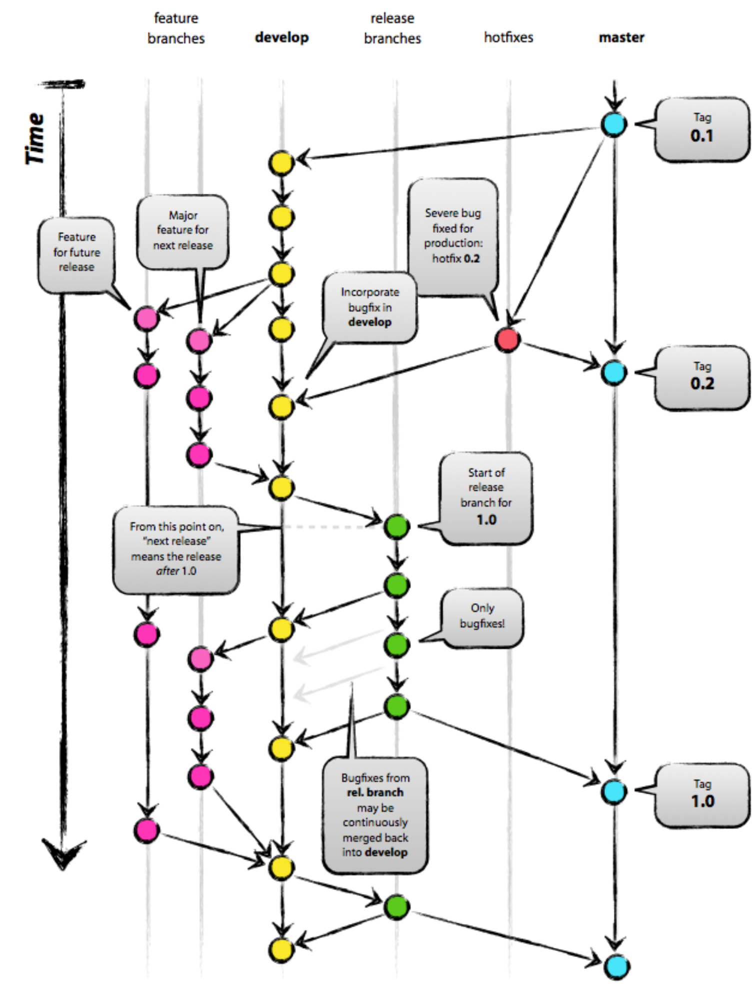
  - Ability to find new bugs in large unannotated code bases (without hand holding)
- Not cost-effective for legacy developers
  - Costly upfront investment
    - Need for specifications
    - Need for environment specifications
    - Need for help with *program-specific* invariants
  - Scalability of (precise) interprocedural analysis
  - Issue of false alarms
    - Users get discouraged after a few “dumb warnings” [Coverity report ‘10]

# Motivation(s)

- How can program verifiers be used by ***any developer*** cost-effectively?
  - Tap (active) research in PL, FM, SE, conferences
  - Answer questions that devs care about (even late in development)
- Does modern software engineering process create **new ways** to apply/leverage/extend program verifiers?

# One direction: differential verification

- Goal
  - Preserve the quality of existing code across evolution (no “regressions”)
- Idea: <u>*Verify properties of program differences*</u>
  - Highlight semantic differences that are unintended
- Research question
  - What properties of differences are interesting?
  - Which of them are amenable to automated verification?
- This talk
  - Some problems in this space
  - Some ongoing solutions



# Motivation: Verifying StringCopy

Need precondition relating dst,src, size, null-terminated

Need a program-specific loop invariant

Check all dereferences are in bound

```
void StringCopy2
(char* dst, char*src, int
size)
{
  int i=0;
  for(;i<size-1 &&
       *src; i++)
    *dst++ = *src++;
  *dst = 0;
}

}
```

# False alarms from no preconditions

size == 2, !Valid(src)
size == 1, !Valid(dst)

```
void StringCopy2
(char* dst, char*src, int
size)
{
  int i=0;
  for(;i<size-1 &&
       *src; i++)
    *dst++ = *src++;
  *dst = 0;
}

}
```

assert(Valid(x))
before every *x

# Weaken the soundness: relative correctness

```c
void StringCopy1
(char* dst, char*src, int
size)
{
  int i=0;
  for(;*src &&
       i<size-1; i++)
    *dst++ = *src++;
  *dst = 0;
}
```

```c
void StringCopy2
(char* dst, char*src, int
size)
{
  int i=0;
  for(;i<size-1 &&
       *src; i++)
    *dst++ = *src++;
  *dst = 0;
}

}
```

Is there any input that passes StringCopy1 but fails StringCopy2?

# Relative correctness (Proof)

```
void StringCopy1
(char* dst, char*src, int
size)
{
  int i=0;
  for(;*src &&
       i<size-1; i++)
    *dst++ = *src++;
  *dst = 0;
}
```

```
void StringCopy2
(char* dst, char*src, int
size)
{
  int i=0;
  for(;i<size-1 &&
       *src; i++)
    *dst++ = *src++;
  *dst = 0;
}
```

No need for any preconditions

Mutual loop invariants:

src.1=src.2, dst.1=dst.2, size.1=size.2, i.1=i.2,

Mem_char.1 == Mem_char.2, ok1 → ok2

# Problems

- Procedure-level equivalence rarely holds for feature-additions, bug-fixes, refactoring
- Equivalence checking for evolving compilers
  - FSE’13, CAV’15
- Differential Assertion Checking and VMV
  - FSE’13, PLDI’14
- Relative bounds and termination
- Semantic Diff for Concurrent Programs
- Semantic Merge
- Semantic Change Impact Analysis

Explored in a tool **SymDiff**

# Problems

- Procedure-level equivalence rarely holds for feature-additions, bug-fixes, refactoring
- Equivalence checking for evolving compilers
  - FSE’13, CAV’15
- **Differential Assertion Checking** and VMV
  - FSE’13, PLDI’14
- Relative bounds and termination
- Semantic Diff for Concurrent Program
- Semantic Merge
- Semantic Change Impact Analysis

Explored in a tool **SymDiff**

# Reduce differential analysis → single program analysis

```
proc f1(x1): r1
modifies g1
{
   s1;
L1:
   w1 := call h1(e1);
   t1
}
```

```
proc f2(x2): r2
modifies g2
{
   s2;
L2:
   w2 := call h2(e2);
   t2
}
```

Novel product construction

```
proc      f1_f2(x1,x2) returns (r1,r2)
modifies g1, g2
{
   // initialize  call  witness  variables
   b_l1 , b_l2 , ... := false , false , ...;

   [[s1 ;]]
L1:
   i_l1 , gi_l1  := e1, g1 ; //store inputs
   call w1 := h1(e1);
   b_l1 := true; //set call witness
   o_l1 , go_l1 := w1, g1; //store outputs

   [[t1 ;]]

   [[s2 ;]]
L2:
   i_l2 , gi_l2  := e2, g2 ; //store inputs
   call w2 := h2(e2);
   b_l2 := true; //set call witness
   o_l2 , go_l2 := w2, g2; //store outputs

   [[t2 ;]]

   //one block for each pair of call sites
   //for a pair of mapped procedures
   ....
   if (b_l1 && b_l2) { //for (L1,L2) pair
      //store the globals
      st_g1 , st_g2 := g1, g2;

      g1, g2 := gi_l1 , gi_l2 ;
      call k1, k2 :=     h1_h2(i_l1, i_l2 );
      assume (k1 == o_l1 && g1 == go_l1);
      assume (k2 == o_l2 && g2 == go_l2);

      //restore globals
      g1, g2 := st_g1 , st_g2 ;
   }
   ...

   return;
}
```

Off-the-shelf program verifier + invariant inference

# Verifying bug fixes

- **Question**: did a fix inadvertently introduce new bugs?
- **Verisec suite**:

  "*snippets of open source programs which contain buffer overflow vulnerabilities, as well as corresponding patched versions*."

  – Examples include apache, madwifi, sendmail, …(~ 50-100 LOC)
  – *Relative memory safety* (buffer overflow) checking
- Automatic proof of relative correctness
  – Using small space of relative invariants {x <= x', x >= x', x == x', x → x', ..}
- Applied similar ideas in **Verification Modulo Versions (VMV)** in CLOUSOT
  – Conditions guaranteeing "bug fix" vs. "regression" (~100KLOC C#)

# Problems

- Procedure-level equivalence rarely holds for feature-additions, bug-fixes, refactoring
- Equivalence checking for evolving compilers
  - FSE’13, CAV’15
- Differential Assertion Checking and VMV
  - FSE’13, PLDI’14
- Relative bounds and termination
- Semantic Diff for Concurrent Program
- **Semantic Merge**
- Semantic Change Impact Analysis

Explored in a tool **SymDiff**

# Semantic merge

Base:

```
int f(int x) {

  return x;
}
```

Variant A:

```
int f(int x) {
  x++;
  return x;
}
```

Variant B:

```
int f(int x) {

  x++;
  return x;
}
```

Incorrect merge:

```
int f(int x) {
  x++;
  x++;
  return x;
}
```

- Inconsistency can be introduced by (text-based) git merge
  - Blamed for Apple SSL/TLS Goto Bug 2014 (led to security

```
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
    goto fail;  /* MISTAKE! THIS LINE SHOULD NOT BE HERE */
```

- Questions
  - Can we have a semantic formulation of conflict-freedom? [revive ’90s work]
  - Can we check such a property automatically?

# Semantic merge

- Verifying conflict freedom for 3-way merge
  - How to represent differences (using *edit scripts)*
  - Formalize conflict-freedom
    - A variable v in **Merge** agrees with the **A** (respectively **B**) if **A** (respectively **B**) changes v's value over **Base**
  - Reduction to assertion checking
  - Sound 4-way product construction
  - Simulation relation inference using Horn Clause Solver (Duality)
- Next step: Semantic merge
  - Synthesize verified merge when git merge fails or causes conflict

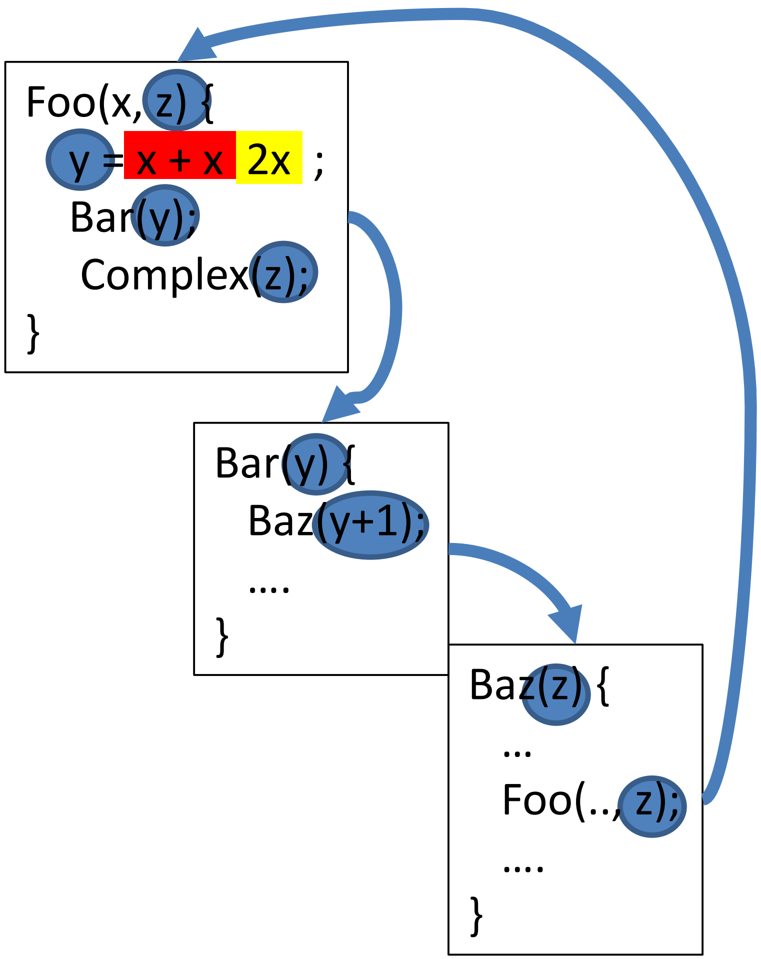
# Problems

  - Procedure-level equivalence rarely holds for feature-additions, bug-fixes, refactoring
- Equivalence checking for evolving compilers
  - FSE’13, CAV’15
- Differential Assertion Checking and VMV
  - FSE’13, PLDI’14
- Relative bounds and termination
- Semantic Diff for Concurrent Program
- Semantic Merge
- **Semantic Change Impact Analysis**

Explored in a tool **SymDiff**

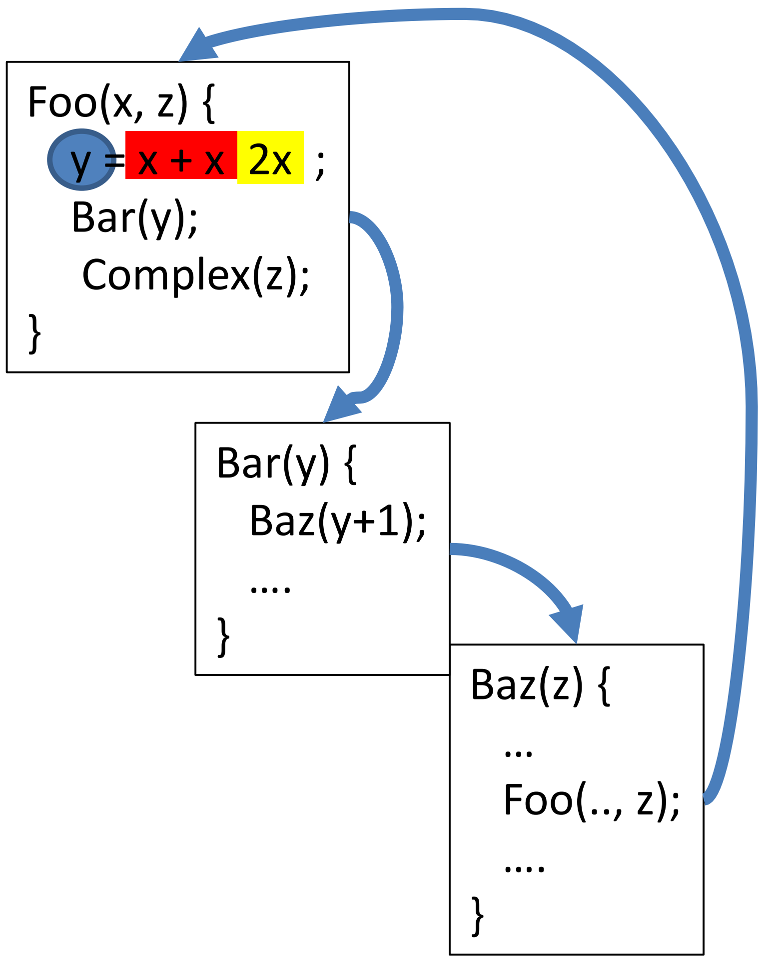
# Semantic change impact

- Problem
  - Which statements are impacted by a change (soundly)?
  - Current approaches don't exploit **change semantics to contain changes**
  - Hard to localize change (even for refactoring parts)
- Solution
  - Incorporate change semantics by inferring **equivalences** when they hold (**SymDiff**)
  - *More subtle than checking two procedures are equal*
  - Novel combination of data-flow and differential invariant inference



# Semantic change impact

- Problem
  - Which statements are impacted by a change (soundly)?
  - Current approaches don't exploit **change semantics to contain changes**
  - Hard to localize change (even for refactoring parts)
- Solution
  - Incorporate change semantics by inferring **equivalences** when they hold (**SymDiff**)
  - More subtle than checking two procedures are equal
  - Novel combination of data-flow and differential invariant inference



# Semantic change impact

- Questions
  - How to formalize CI soundly, not dependent on syntactic diff
  - What kind of semantic/relative facts can help prune impact
  - How to leverage relative verification in a scalable manner with a lightweight static analysis
- Applied it to several GitHub projects using SMACK +SymDiff

# Summary

- Differential verification
  - Verify properties of difference (2+ programs) as opposed to a single program
  - New domain of problems to apply verification
  - Less reliance of specifications, environment modeling and program-specific invariants
- Use cases in software engineering
  - High quality detection of **regressions** (e.g. relative memory safety)
  - Help with **refactorings** (equivalence checking, ..)
  - **Code review** (understand change impact)
  - **Redundant tests** (that only cover non-impacted statements)
  - Safer **merge** (avoid cost regression and rollback later)
  - **Verifying approximations** in compilers (relative assertion, termination safety)
- A new cost-effective way to use automatic verification!

**SymDiff** http://research.microsoft.com/en-us/projects/symdiff/