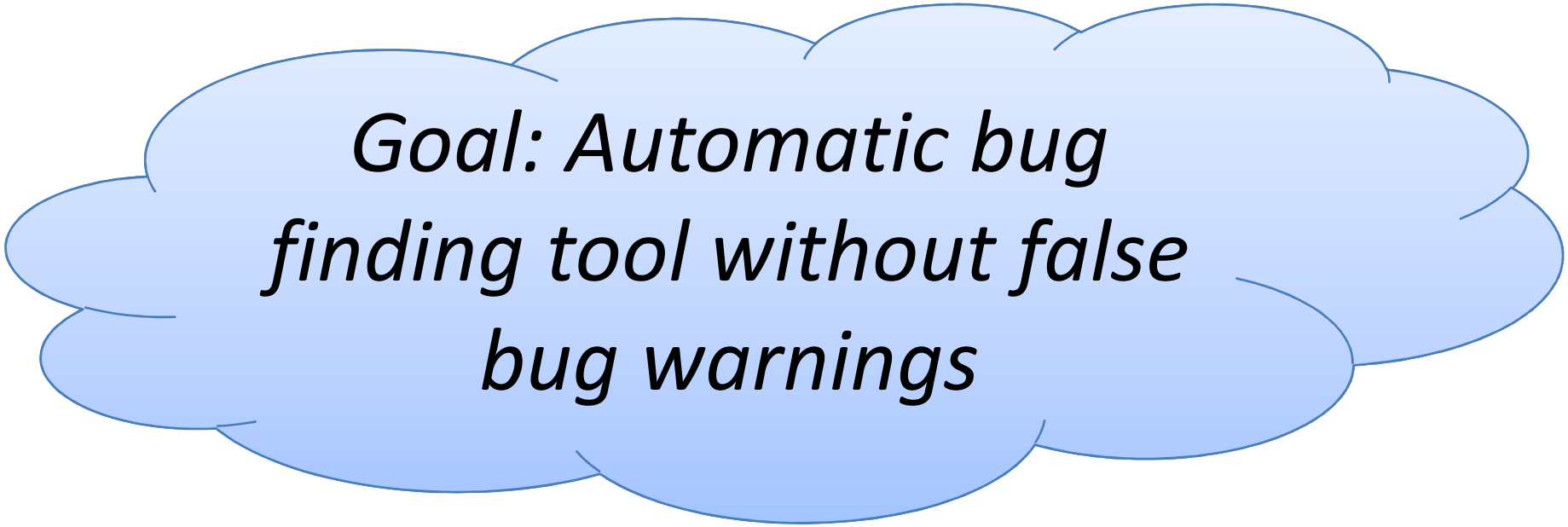
Combining Static and Dynamic Analysis for Bug Finding

Christoph Csallner
Professor Yannis Smaragdakis

18 October 2007

Georgia Tech


Goal: Automatic bug finding tool without false bug warnings

Combining Static and Dynamic Analysis for Bug Finding

Program Analysis for mass-market

(+) Fully automated

Combining Static and Dynamic Analysis for Bug Finding

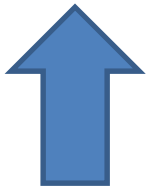
```
/**
 * @param p should be 1 or greater
 */
```

Analyse program and specification

(-) Hard: spec mostly informal

(+) Abstract to analyze all paths

(-) Even impossible ones



Combining Static and Dynamic Analysis for Bug Finding

Model with superset of possible paths

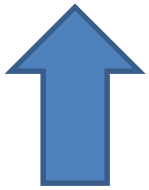
→ Example: compiler

Warn about ``bug'' on impossible path

→ False bug warning

(+) Run only possible paths

(-) Cannot run all paths



Combining Static and Dynamic Analysis for Bug Finding

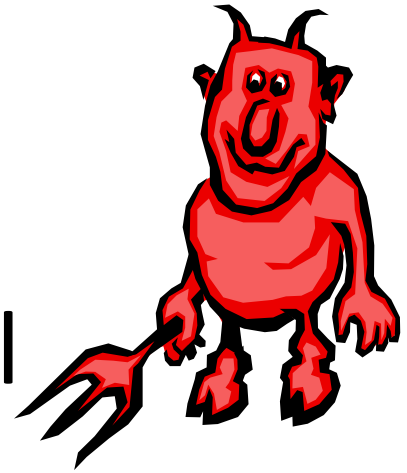
Real bug on missed path

→ Missed bug

Combining Static and Dynamic Analysis for Bug Finding

People have limited time

→ False bug warnings are evil



Many bug warnings, lot of noise

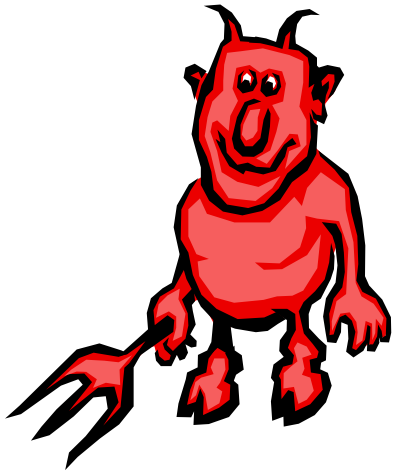
Georgia Tech

False bug warnings: Show-stopper

- Flanagan et al. (ESC/Java people), 2002:
  - "[T]he tool has not reached the desired level of cost effectiveness. In particular, users complain about an annotation burden that is perceived to be heavy, and about **excessive warnings about non-bugs**, particularly on unannotated or partially-annotated programs."
- Rutar et al., 2004:
  - > 9k NPE warnings in 170k non commented source stmt
  - "[..] **too many warnings to be easily useful by themselves**."

Two kinds of false bug warnings

- *Language-level,* e.g.: Java semantics
  - E.g.: `if (0==1) {infeasible};`
  - Behavior cannot occur
  - Never ever



- *User-level*
  - E.g.: passing -1 into `m(int p) //p pos!`
  - Behavior can occur, but user does not care
  - We cannot read minds, but worth trying

Goal: Automatic bug finding tool without false bug warnings

Huge, unsolved problem

(+) Commercial interest

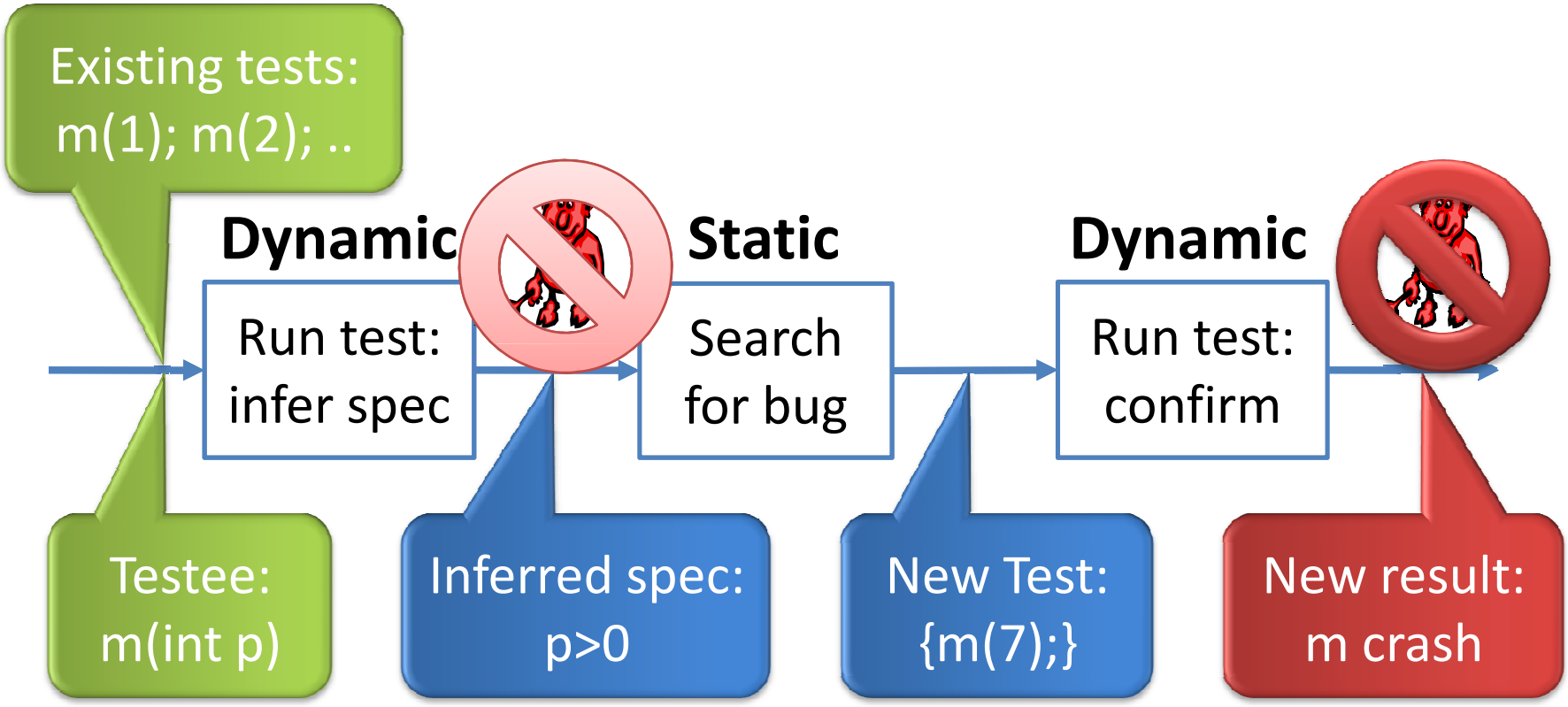
Sound(bug-finder) = Complete(correctness-prover)

- *Sound*: claim(p) $\Rightarrow$ p
  - Bug finding: No false bug warnings
  - Prove correctness: Find all bugs

- *Complete*: p $\Rightarrow$ claim(p)
  - Bug finding: Find all bugs
  - Prove correctness: No false bug warnings
- Sound(bug-finding) = Complete(prove correctness)
- Sound(prove correctness) = Complete(bug-finding)

Our tool-chain: DSD-Crasher

ISSTA 2006 paper: ``DSD-Crasher'' **[distinguished paper award]**
(ACM SIGSOFT International Symposium on Software Testing and Analysis)



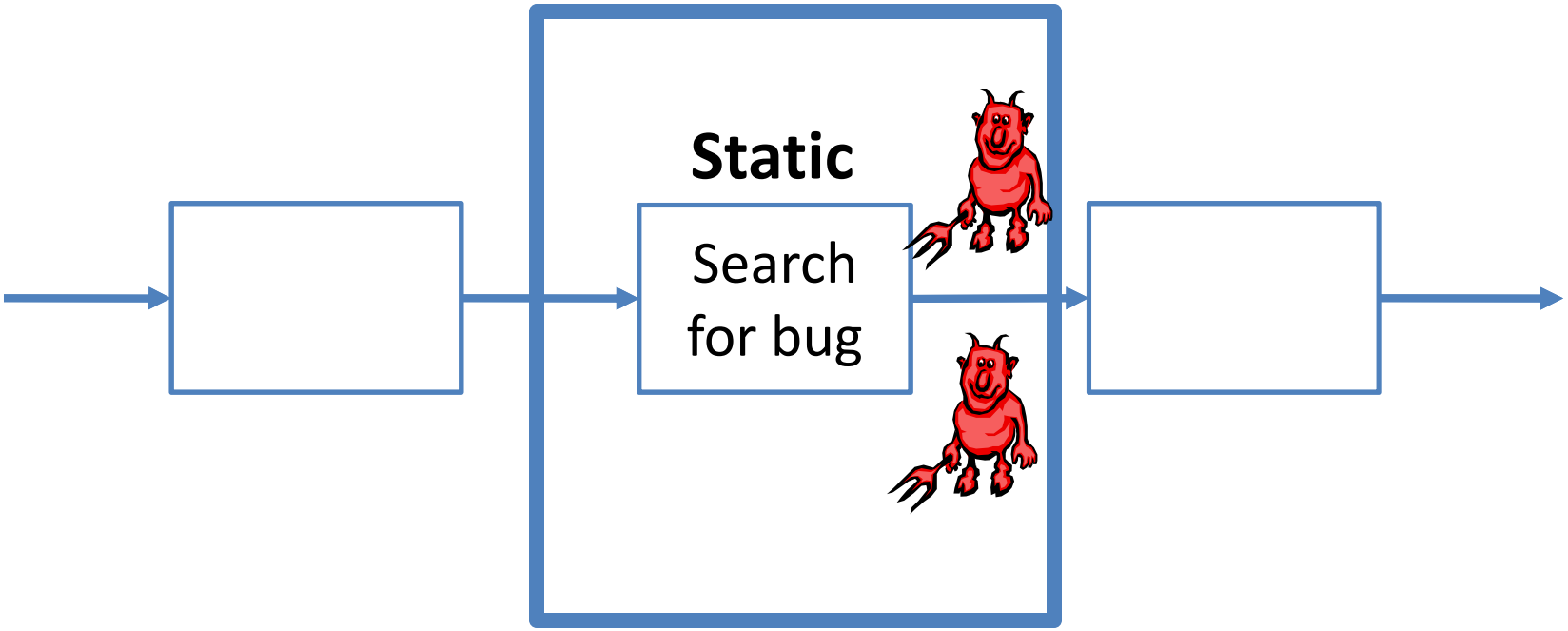
Remove user-level false warnings

Core Search

Remove language-level false warnings

ESC/Java: Static Analysis
Core bug search component

Cormac Flanagan et al. (Compaq)



Core Search

ESC/Java: Static Analysis
Core bug search component

- Checks for potential invariant violations
  - User-defined in JML [Gary T. Leavens]
  - **Pre**conditions, **Post**conditions, class invariants
- Knows pre and post of primitive language ops
  - Pointer dereference, class cast, array access, etc.
- Checks each method in isolation
  - Call → check pre, assume post
  - Method body → its weakest pre

ESC under the hood

- Weakest pre **`wp(method body, true)`**
  - States from which execution terminates normally
  - Remaining states lead execution to an error: Violate some pre or post
- We are interested in those that throw a runtime exception
  - Dereferencing null, Illegal type cast, Illegal array allocation or access, Division by zero
- ESC uses Simplify to derive abstract error

Expected static analysis problem: False bug warnings

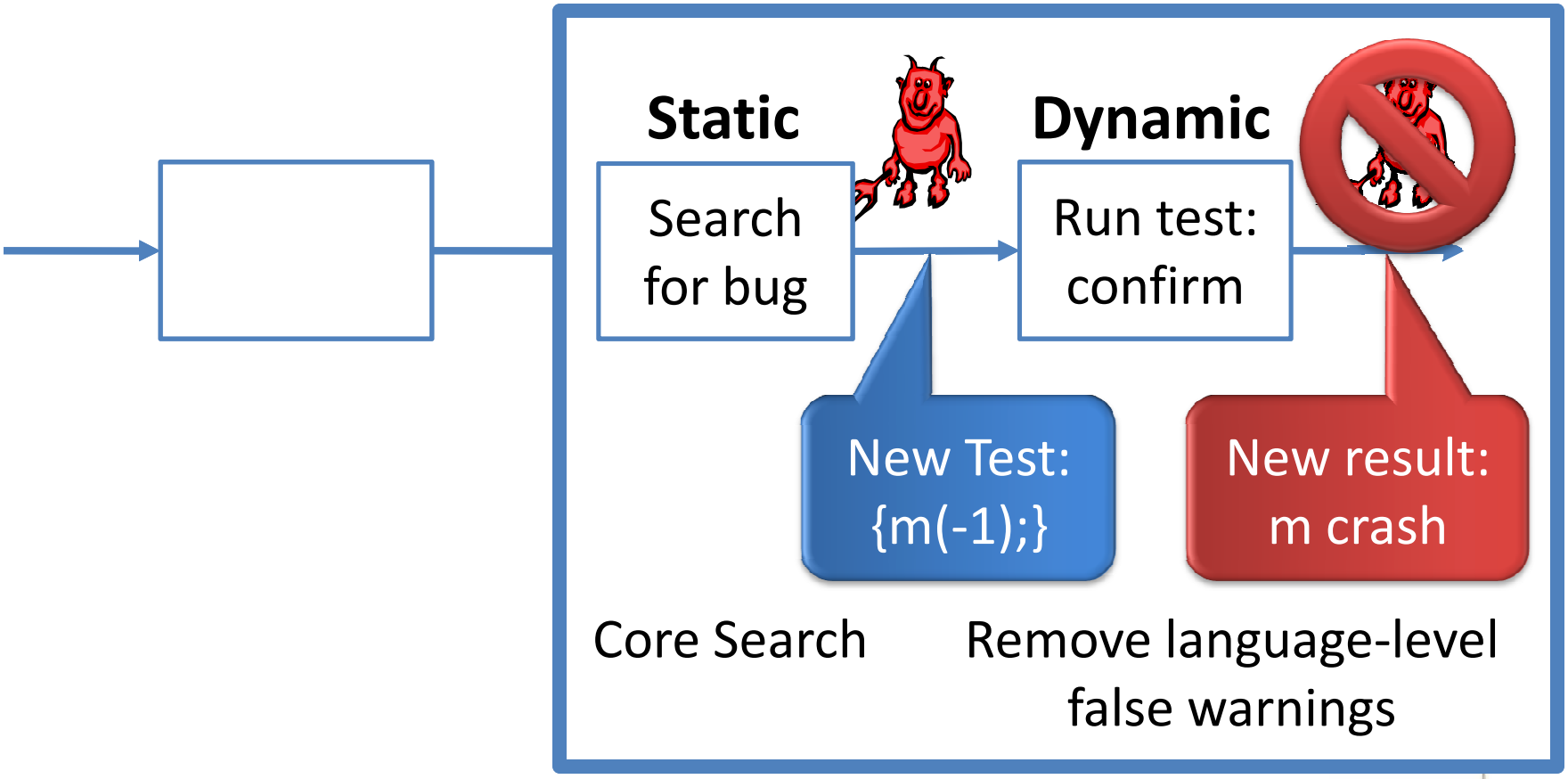
- Example:
- `public int get10() {return 10;}`
-
```
public int meth(int p) {
    return p / get10();
}
```
- ESC warns of a division by zero in meth
  - Can never occur!

Less expected: Output cryptic

(arrayLength(firstArray:294.43) <= intLast)
(firstArray:294.43 < longFirst)
(tmp0!new!double[]:297.27 < longFirst)
(secondArray:295.43 < longFirst)
(longFirst < intFirst)
(vAllocTime(tmp0!new!double[]:297.27) < alloc<1>)
(0 < arrayLength(firstArray:294.43))
(null <= max(LS))
(vAllocTime(firstArray:294.43) < alloc)
(alloc <= vAllocTime(tmp0!new!double[]:297.27))
(eClosedTime(elems@loopold) < alloc)
(vAllocTime(out:6..) < alloc)
(vAllocTime(secondArray:295.43) < alloc)
(intLast < longLast)
(1000001 <= intLast)
((intFirst + 1000001) <= 0)
(out:6.. < longFirst)
arrayLength(tmp0!new!double[]:297.27) ==
  arrayLength(firstArray:294.43)
typeof(0) <: T_int
null.LS == @true
typeof(firstArray:294.43[0]) <: T_double
isNewArray(tmp0!new!double[]:297.27) == @true
typeof(arrayLength(firstArray:294.43)) <: T_int
T_double[] <: arrayType
typeof(firstArray:294.43) == T_double[]
T_double[] <: T_double[]
elemtype(T_double[]) == T_double
T_double <: T_double
typeof(secondArray:295.43) <: T_double[]
typeof(secondArray:295.43) == T_double[]
arrayFresh(tmp0!new!double[]:297.27 alloc alloc<1>
  elems@loopold
arrayShapeOne(arrayLength(firstArray:294.43))
  T_double[] F_0.0)
typeof(firstArray:294.43) <: T_double[]
typeof(tmp0!new!double[]:297.27) == T_double[]
(null <= max(LS))
tmp0!new!double[]:297.27[0] == firstArray:294.43[0]
arrayLength(secondArray:295.43) == 0
elems@loopold == elems
elems@pre == elems
state@pre == state
state@loopold == state
m@loopold:299.12 == 0
out@pre:6.. == out:6..
EC@loopold == EC
alloc@pre == alloc
!typeof(out:6..) <: T_float
!typeof(out:6..) <: T_byte
!typeof(secondArray:295.43) <: T_boolean
!typeof(firstArray:294.43) <: T_double
!typeof(secondArray:295.43) <: T_short
!typeof(tmp0!new!double[]:297.27) <: T_boolean
!typeof(tmp0!new!double[]:297.27) <: T_short
!typeof(firstArray:294.43) <: T_boolean
!typeof(out:6..) <: T_char
!typeof(secondArray:295.43) <: T_double
!typeof(firstArray:294.43) <: T_float
!typeof(out:6..) <: T_int
!isAllocated(tmp0!new!double[]:297.27 alloc)
!typeof(secondArray:295.43) <: T_long
!typeof(firstArray:294.43) <: T_char
!typeof(tmp0!new!double[]:297.27) <: T_double
!typeof(tmp0!new!double[]:297.27) <: T_long
T_double[] != T_boolean
T_double[] != T_char
T_double[] != T_byte
T_double[] != T_long
T_double[] != T_short
T_double[] != T_int
T_double[] != T_float
!typeof(firstArray:294.43) <: T_byte
!typeof(out:6..) <: T_boolean
!typeof(secondArray:295.43) <: T_float
!typeof(out:6..) <: T_short
!typeof(secondArray:295.43) <: T_byte
!typeof(tmp0!new!double[]:297.27) <: T_float
!typeof(firstArray:294.43) <: T_long
!typeof(tmp0!new!double[]:297.27) <: T_byte
!typeof(out:6..) <: T_double
!typeof(firstArray:294.43) <: T_short
!typeof(out:6..) <: T_long
typeof(out:6..) != T_boolean
typeof(out:6..) != T_char
typeof(out:6..) != T_byte
typeof(out:6..) != T_long
typeof(out:6..) != T_short
typeof(out:6..) != T_int
typeof(out:6..) != T_float
!typeof(secondArray:295.43) <: T_char
!typeof(secondArray:295.43) <: T_int
!typeof(tmp0!new!double[]:297.27) <: T_char
!typeof(firstArray:294.43) <: T_int
!typeof(tmp0!new!double[]:297.27) <: T_int
bool$false != @true
secondArray:295.43 != null
firstArray:294.43 != null
T_double != typeof(out:6..)
T_double != T_double[]
tmp0!new!double[]:297.27 != null

Check ‘n’ Crash: Language-level sound bug warnings

ICSE 2005 paper: ``Check ‘n’ Crash’’
(ACM/IEEE International Conference on Software Engineering)



Finding test cases that satisfy ESC's error conditions

- ESC uses Simplify automated theorem prover
  - Limitations: multiplication, float, large numbers, …
- We use POOC int solver [Schlenker et al.]
- float and double: currently ignored
- Object reference: Keep equivalence relation
- Array: Reference to a special object
  - Int constraints on array length and indices
- Rest: JCrasher random test case generator

Filling in random values: JCrasher

Software, Practice & Experience 2004 paper: ``JCrasher''

- Predefined values, e.g.:

```
int ← 0
int ← 1
A ← null
```

- Crawl testee for method signatures:

```
A ← (B,int)
```
//from method `A B.m(int)`

- Chain discovered rules recursively:
  - `(new B()).m(0)`
  - `(new B()).m(1)`
  - …

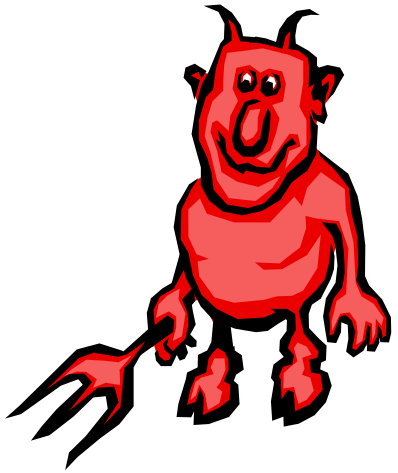
Example: Student homework

- ```
  public static void swapArrays
      (double[] fstArray, double[] sndArray) { //..
    for(int m=0; m<fstArray.length; m++) { //..
      fstArray[m]=sndArray[m]; /*..*/ } }
  ```
- Informal spec:
  - Swap `fstArray` and `sndArray` elements
  - If array lengths differ, just return
- Corresponding ESC/Java warning:

```
Array index possibly too large (IndexTooBig)
    fstArray[m]=sndArray[m];
                        ^
```
```

Example: ESC's counterexample

(arrayLength(firstArray:294.43) <= intLast)
(firstArray:294.43 < longFirst)
(tmp0!new!double[]:297.27 < longFirst)
(secondArray:295.43 < longFirst)
(longFirst < intFirst)
(vAllocTime(tmp0!new!double[]:297.27) < alloc<1>)
(0 < arrayLength(firstArray:294.43))
(null <= max(LS))
(vAllocTime(firstArray:294.43) < alloc)
(alloc <= vAllocTime(tmp0!new!double[]:297.27))
(eClosedTime(elems@loopold) < alloc)
(vAllocTime(out:6..) < alloc)
(vAllocTime(secondArray:295.43) < alloc)
(intLast < longLast)
(1000001 <= intLast)
((intFirst + 1000001) <= 0)
(out:6.. < longFirst)
arrayLength(tmp0!new!double[]:297.27) ==
arrayLength(firstArray:294.43)
typeof(0) <: T_int
null.LS == @true
typeof(firstArray:294.43[0]) <: T_double
isNewArray(tmp0!new!double[]:297.27) == @true
typeof(arrayLength(firstArray:294.43)) <: T_int
T_double[] <: arrayType
typeof(firstArray:294.43) == T_double[]
T_double[] <: T_double[]
elemtype(T_double[]) == T_double
T_double <: T_double
typeof(secondArray:295.43) <: T_double[]
typeof(secondArray:295.43) == T_double[]
arrayFresh(tmp0!new!double[]:297.27 alloc alloc<1>
elems@loopold
arrayShapeOne(arrayLength(firstArray:294.43))
T_double[] F_0.0)
typeof(firstArray:294.43) <: T_double[]
typeof(tmp0!new!double[]:297.27) == T_double[]
(null <= max(LS))
tmp0!new!double[]:297.27[0] == firstArray:294.43[0]
arrayLength(secondArray:295.43) == 0
elems@loopold == elems
elems@pre == elems
state@pre == state
state@loopold == state
m@loopold:299.12 == 0
out@pre:6.. == out:6..
EC@loopold == EC
alloc@pre == alloc
!typeof(out:6..) <: T_float
!typeof(out:6..) <: T_byte
!typeof(secondArray:295.43) <: T_boolean
!typeof(firstArray:294.43) <: T_double
!typeof(secondArray:295.43) <: T_short
!typeof(tmp0!new!double[]:297.27) <: T_boolean
!typeof(tmp0!new!double[]:297.27) <: T_short
!typeof(firstArray:294.43) <: T_boolean
!typeof(out:6..) <: T_char
!typeof(secondArray:295.43) <: T_double
!typeof(firstArray:294.43) <: T_float
!typeof(out:6..) <: T_int
!isAllocated(tmp0!new!double[]:297.27 alloc)
!typeof(secondArray:295.43) <: T_long
!typeof(firstArray:294.43) <: T_char
!typeof(tmp0!new!double[]:297.27) <: T_double
!typeof(tmp0!new!double[]:297.27) <: T_long
T_double[] != T_boolean
T_double[] != T_char
T_double[] != T_byte
T_double[] != T_long
T_double[] != T_short
T_double[] != T_int
T_double[] != T_float
!typeof(firstArray:294.43) <: T_byte
!typeof(out:6..) <: T_boolean
!typeof(secondArray:295.43) <: T_float
!typeof(out:6..) <: T_short
!typeof(secondArray:295.43) <: T_byte
!typeof(tmp0!new!double[]:297.27) <: T_float
!typeof(firstArray:294.43) <: T_long
!typeof(tmp0!new!double[]:297.27) <: T_byte
!typeof(out:6..) <: T_double
!typeof(firstArray:294.43) <: T_short
!typeof(out:6..) <: T_long
typeof(out:6..) != T_boolean
typeof(out:6..) != T_char
typeof(out:6..) != T_byte
typeof(out:6..) != T_long
typeof(out:6..) != T_short
typeof(out:6..) != T_int
typeof(out:6..) != T_float
!typeof(secondArray:295.43) <: T_char
!typeof(secondArray:295.43) <: T_int
!typeof(tmp0!new!double[]:297.27) <: T_char
!typeof(firstArray:294.43) <: T_int
!typeof(tmp0!new!double[]:297.27) <: T_int
bool$false != @true
secondArray:295.43 != null
firstArray:294.43 != null
T_double != typeof(out:6..)
T_double != T_double[]
tmp0!new!double[]:297.27 != null

Process ESC's counterexample

- Feed relevant constraints to constraint solvers
- Compile solutions into a JUnit test case:
- ```
  public void test0() throws Throwable {
      try {
              double[] d1 = new double[]{1.0};
              double[] d3 = new double[]{};
              P1s1.swapArrays(d1, d3);
      } catch (Exception e) {dispatchException(e);}
  }
  ```
- Observe test case execution
  - Use JCrasher's extensions of JUnit runtime


  ```

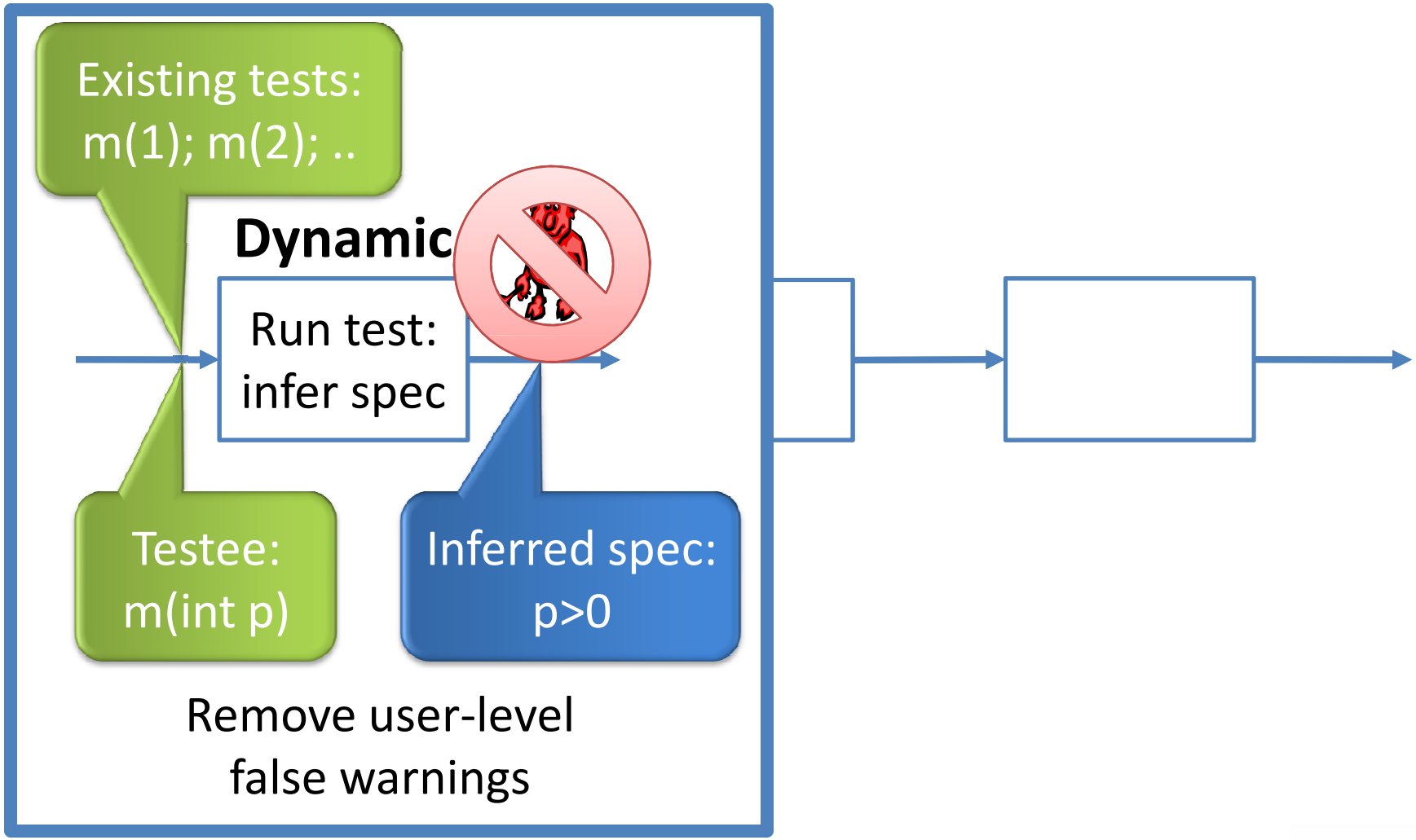
Two kinds of false bug warnings

- *Language-level,* e.g.: Java semantics
  - E.g.: `if (false) {infeasible};`
  - Behavior cannot occur
  - Never ever



- *User-level*
  - E.g.: passing -1 into `m(int p) //p pos!`
  - Behavior can occur, but user does not care
  - We cannot read minds, but worth trying

Front-end: User-level soundness



Daikon: Infer invariants

Michael Ernst et al. (MIT)

1. Instrument testee bytecode, execute
   - Trace variable values at method entry & exit
2. Analyze each entry/exit value set
   - Instantiate invariant templates with values
   - Invariant invalidated by sample → Drop invariant
   - Invariant held for some samples, never invalidated → Assume: true invariant
3. Annotate testee's source code (with JML):
   - Preconditions, postconditions, class invariants

Daikon: Configuration

- Concentrate on simple invariants, e.g.
  - intVariable {=,≥,>, ≤,<} {intConstant, intVariable}
- Ignore complex invariants, benefit unclear:
  - variable is one of {const1, const2, ..}
  - Elements of container structures
  - float, double, String

Experiments Overview

- ESC/Java produces many false bug warnings
- Check ‘n’ Crash on its own discovers real bugs
  - In JBoss JMS, JABA, Groovy, student homeworks
- DSD-Crasher improves soundness
  - Fewer false bug reports
- DSD-Crasher improves on other tools with less sophisticated static analysis
  - Finds more bugs

Improvement over Check ‘n’ Crash: Groovy experiments

- Use test suite that comes with Groovy

Tool	Runtime [min:s]	Exception reports
Check ‘n’ Crash	10:43	19
DSD-Crasher	30:32	11

- Using Daikon-inferred invariants
  - 12..18: ESC could statically **rule out false positives**
  - 19: ESC produces more complicated error condition, threw off constraint solver

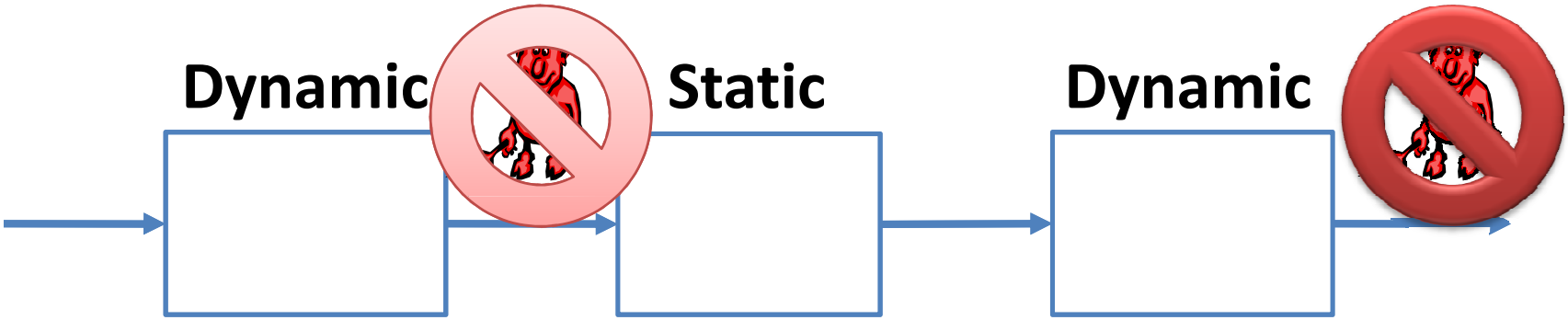
Example improvement over Check 'n' Crash

- JBoss JMS: Check 'n' Crash warns of NegativeArraySizeException
-
```
setBytes(byte[] value, int length) //simplified
{
  byte[] bytes = new byte[length];  //..
}
```
- Test case calls **setBytes** three times
- Daikon infers precondition:

  length == daikon.Quant.size(value)
- DSD-Crasher suppresses false positive

Eclat: DSD combination with less static analysis

Carlos Pacheco et al. (MIT)



Remove user-level false warnings

Core Search

Remove language-level false warnings

DSD-Crasher: Deeper than Eclat

- From Groovy experiments
- DSD-Crasher benefits from deeper static analysis
- Eclat does more than we look for

Tool (timeout)	Exception reports	Runtime [min:s]
Eclat-default	0	7:01
Eclat-hybrid, 4 rounds	0	8:24
Eclat-exh, 2 rounds	2	10:02
Eclat-exh (500 s)	2	16:42
Eclat-exh (1200 s)	2	33:17
DSD-Crasher	**4**	**30:32**

(Eclat-exh = Eclat's exhaustive search)

DSD-Crasher: Example Improvement Over Eclat

- Eclat is a Daikon+test generation combination
  - Static analysis mostly random
  - Does not find following bug for any settings
- JBoss JMS example
  - **byte[] getBytes(Object value) //simplified**

```
{
  if (value == null) {return null;}
  else if (value instanceof Byte[]) {
    return (byte[]) value;
  } //..
}
```

Selected related bug-finding work

- Dynamic (JCrasher)
  - Boyapati et al. (ISSTA ‘02), Pacheco et al. (ICSE ‘07)
- Static-Dynamic (Check ‘n’ Crash)
  - Beyer et al. (ICSE ‘04), Tomb et al. (ISSTA ‘07)
- Dynamic-Static-Dynamic (DSD-Crasher)
  - Xie et al. (ASE ‘03), Pacheco et al. (ECOOP ‘05)
- Concolic (Static and dynamic in parallel)
  - Godefroid et al. (PLDI ‘05), Sen et al. (FSE ‘05)

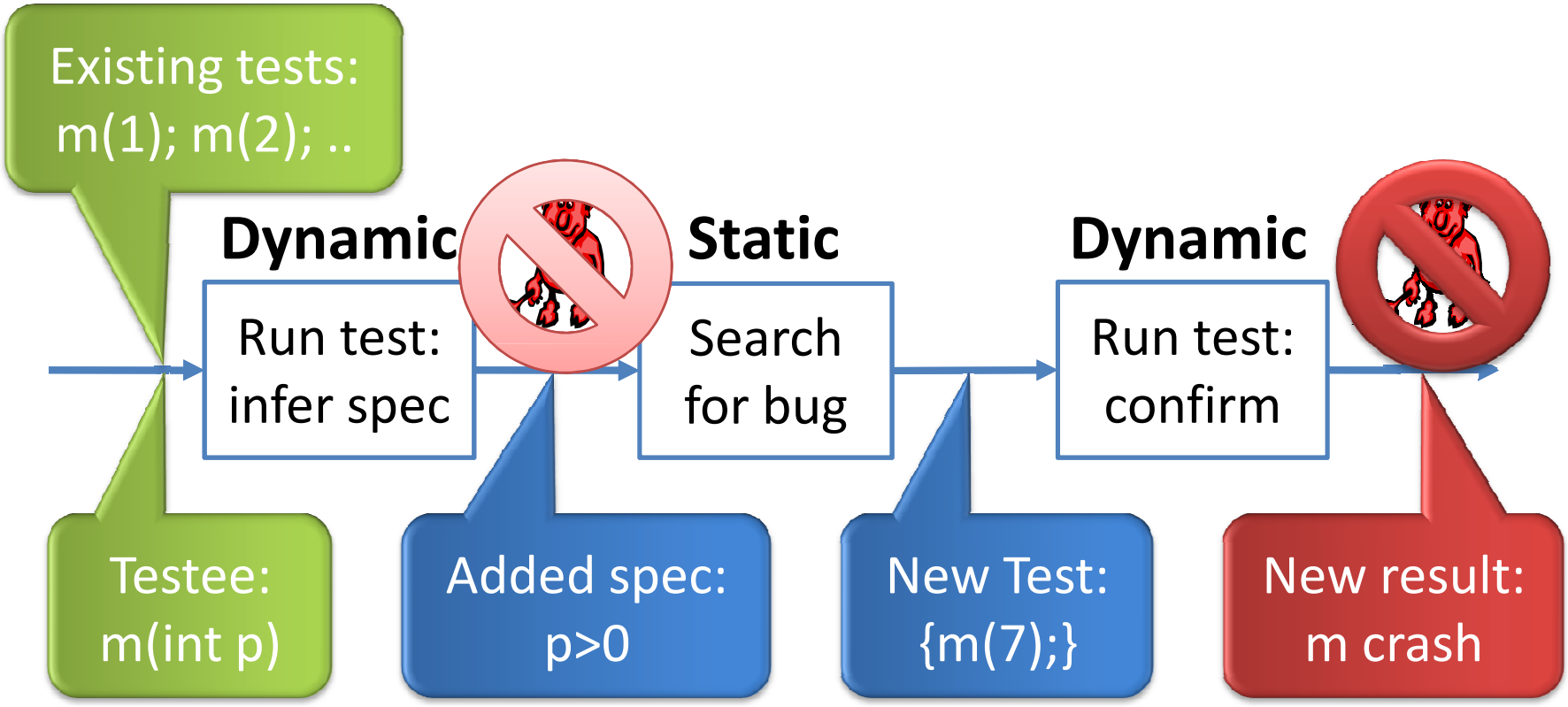
Our tools are used

- Check ’n’ Crash
  - Freie Universität Berlin: Lab course
  - North Carolina State University
    - Sarah Smith Heckman et al.: Rank output of bug finding tools (Short papers in ISSRE ‘05, ISSRE ’06, ICSE ‘07)
  - UC Santa Cruz & RIACS/NASA Ames
    - Aaron Tomb et al.: Compare with advanced tool (ISSTA ‘07)
- JCrasher in teaching:
    - Drexel University: Homework assignment (SE320)
    - Universiteit Utrecht: Seminar (Program Verification)

Examples of JCrasher in research

- University of Washington: David Notkin's group
  - Characterize test cases (ICFEM '04, ISSRE '05)
- North Carolina State University: Tao Xie's group
  - Minimize random tests (ASE '04)
  - Add assertions to random tests (ECOOP '06)
  - Web service robustness (ICWS '07)
- MIT: Michael Ernst's group
  - Compare with Eclat (ECOOP '05)
  - With Microsoft Research: Randoop comparison (ICSE '07)

Our tool-chain: DSD-Crasher
Dynamic-Static-Dynamic



Remove user-level false warnings

Core Search

Remove language-level false warnings

Georgia Tech